

VŠB - Technická univerzita Ostrava  
Fakulta elektrotechniky a informatiky  
Katedra informatiky

Platforma pro výuku vybraných partií  
matematiky a informatiky

Platform for Selected Parts of Mathematics  
and Computer Science Teaching

VŠB - Technická univerzita Ostrava  
Fakulta elektrotechniky a informatiky  
Katedra informatiky

## Zadání diplomové práce

Student:

**Bc. Libor Polehňa**

Studijní program:

N2647 Informační a komunikační technologie

Studijní obor:

2612T025 Informatika a výpočetní technika

Téma:

Platforma pro výuku vybraných partií matematiky a informatiky  
Platform for Selected Parts of Mathematics and Computer Science  
Teaching

Jazyk vypracování:

čeština

Zásady pro vypracování:

Cílem práce je vytvoření systému pro výuku matematiky a opakování získaných informací. Pro většinu specifických oblastí vysokoškolské matematiky a informatiky neexistuje software, který by student mohl použít pro ověření svých výsledků nebo názorně zobrazit postup samotného řešení úlohy. Zadáním je vyvinout platformu, která umožní práci se specifickými vstupy pro různé oblasti matematiky a informatiky, zobrazí postup řešení a nabídne materiál ke studiu.

1. Analyzujte stávající řešení, identifikujte klíčové nedostatky a navrhnete nové funkce, které tyto nedostatky řeší.
2. Analyzujte navrhované řešení, sbírejte požadavky, nalezněte klíčové případy užití.
3. Navrhnete architektury řešení s důrazem na flexibilitu řešení ve vztahu k přidání nových funkcí a změnám použitých technologií.
4. Implementujte systém. Implementace je klíčovou částí práce a proto bude kladen důraz na funkčnost řešení a dodržení dobrých praktik programování.

Seznam doporučené odborné literatury:

- [1] ZAKI, Mohammed J.; MEIRA JR, Wagner; MEIRA, Wagner. Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press, 2014.
- [2] ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Přeložil Bogdan KISZKA. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.
- [3] MELOUN, Milan a Jiří MILITKÝ. Kompendium statistického zpracování dat. Vyd. 3. Praha: Karolinum, 2012. ISBN 978-80-246-2196-8.
- [4] CHAMBERS, John M. Graphical Methods for Data Analysis: 0. Chapman and Hall/CRC, 2017.
- [5] VÁVRŮ, Jiří. iPhone: vývoj aplikací. Praha: Grada Publishing, 2012. ISBN 978-80-247-4457-5.
- [6] LUCIE, Rohlíková a JANA, Vejvodová. Vyučovací metody na vysoké škole: Praktický průvodce výukou v prezenční i distanční formě studia. Grada Publishing as, 2012.

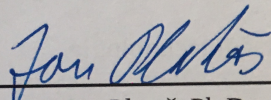
V Ostravě 26. dubna 2019

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

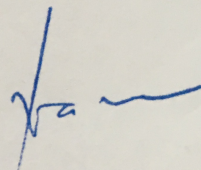
Vedoucí diplomové práce: **Ing. Jana Nowaková, Ph.D.**

Datum zadání: 01.09.2018

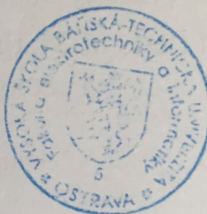
Datum odevzdání: 30.04.2019



doc. Ing. Jan Platoš, Ph.D.  
vedoucí katedry



prof. Ing. Pavel Brandštetter, CSc.  
děkan fakulty






## Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2019

.....

## Poděkování

Chtěl bych poděkovat své vedoucí diplomové práce Ing. Janě Nowakové, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

## **Abstrakt**

Cílem diplomové práce je vytvořit platformu pro podporu výuku vybraných partií matematiky a teoretické informatiky. Navržený systém je schopen poskytnout uživateli možnost provádět matematické výpočty a operace z uživatelského rozhraní, bez znalosti programování. Uživatel má možnost vidět postup řešení výpočtu nebo operace. Představené uživatelské rozhraní je jednodušší a přehlednější v porovnání s obdobnými řešeními dostupných na platformě iOS v současné době. Hlavním zaměřením navrženého systému je pokrytí partií, které nelze v existujících softwarech moc najít. Celý systém je složen z mobilní aplikace pro operační systém iOS a serverového backendu. Práce obsahuje analýzu současného stavu řešení, sběr požadavků, návrh řešení, teoretický popis a implementaci.

## **Klíčová slova**

analýza dat, algebra, teoretická informatika, matematika, výuka, iOS, mobilní aplikace, vývoj software

## **Abstract**

The objective of master thesis is to create platform for teaching of selected parts of mathematics and computer science. System is able to provide possibility to user to make a mathematics calculations and operations from user interface, without knowledge of programming. User is able to see a step by step solution of calculation and operation of any math problem. System provides easier and clearer user interface than existing systems on iOS platform. The main aim of system is covering a mathematics parts which are hard to find in existing systems. The whole system is consists of mobile application for iOS operating system and server backend. Thesis includes analysis of existing systems, requirements gathering, solution design, theoretical description and implementation.

## **Keywords**

data analysis, algebra, computer science, mathematics, teaching, iOS, mobile application, software development

# Obsah

<b>1</b>	<b>Úvod</b>	<b>14</b>
<b>2</b>	<b>Analýza současného stavu řešení</b>	<b>17</b>
2.1	Apple iOS	17
2.1.1	Photomath	17
2.1.2	Math 42	17
2.1.3	iMathematics	18
2.1.4	FX Math Solver	18
2.1.5	Symbolab Calculator	18
2.1.6	Mathway	19
2.1.7	Khan Academy	19
2.1.8	Brilliant	20
2.2	Desktop a Web	20
2.2.1	Wolfram Mathematica a Wolfram Alpha	20
2.2.2	Matlab	20
2.3	Shrnutí	20
<b>3</b>	<b>Návrh výpočetního modelu</b>	<b>22</b>
3.1	Teoretický základ	22
3.1.1	Matice	22
3.1.1.1	Definice	23
3.1.1.2	Speciální typy	23
3.1.1.3	Operace	24
3.1.1.4	Základní úpravy	24
3.1.2	Regulární jazyky	25
3.1.2.1	Definice	25
3.1.2.2	Operace	26
3.1.3	Konečné automaty	26
3.1.3.1	Definice - Deterministický konečný automat	26
3.1.3.2	Definice - Nedeterministický konečný automat	27
3.1.3.3	Zásobníkový automat	27
3.1.4	Bezkontextové gramatiky	28
3.1.4.1	Definice	28
3.1.4.2	Derivace	28
3.1.4.3	Ukázkový příklad	29
3.1.5	Machine learning	30
3.1.5.1	Základní typy učení	30
3.1.5.2	Základní typy úloh	31
3.1.5.3	K-means	31
3.1.6	Evoluční algoritmy	33

3.1.6.1	Genetický algoritmus . . . . .	34
3.1.7	Teorie grafů . . . . .	35
3.1.7.1	Definice . . . . .	36
3.1.7.2	Základní vlastnosti a typy grafů . . . . .	36
3.1.8	Game of Life . . . . .	37
3.1.8.1	Pravidla . . . . .	37
3.2	Funkce, operace . . . . .	38
3.2.1	Maticе . . . . .	38
3.2.1.1	Gaussova eliminační metoda . . . . .	38
3.2.1.2	Gauss-Jordánová eliminační metoda . . . . .	39
3.2.2	Regulární jazyk - převod na nedeterministický konečný automat . . . . .	39
3.2.2.1	Primitivní základní automaty . . . . .	39
3.2.2.2	Automat pro zřetězení . . . . .	40
3.2.2.3	Automat pro sjednocení . . . . .	40
3.2.2.4	Automat pro iteraci . . . . .	41
3.2.3	Konečný automat . . . . .	42
3.2.3.1	Převod na deterministický konečný automat . . . . .	42
3.2.3.2	Minimalizace . . . . .	43
3.2.3.3	Simulace . . . . .	46
3.2.4	Bezkontextové gramatiky - redukce . . . . .	46
3.2.4.1	Odstranění neterminálů, které negenerují terminální slovo . . . . .	47
3.2.4.2	Odstranění nedosažitelných neterminálů . . . . .	47
3.2.5	Machine learning - K-means . . . . .	48
3.2.6	Evoluční algoritmy - Genetický algoritmus . . . . .	49
3.2.7	Teorie grafů . . . . .	50
3.2.7.1	Vykreslení grafu . . . . .	51
3.2.7.2	Animace . . . . .	52
3.3	Syntaktická analýza . . . . .	53
3.3.1	Překlad shora dolu . . . . .	54
3.3.2	First a Follow . . . . .	54
3.3.2.1	FIRST . . . . .	54
3.3.2.2	FOLLOW . . . . .	55
3.3.2.3	Příklad . . . . .	56
3.3.3	LL(1) gramatiky . . . . .	56
3.3.4	Parser . . . . .	57
3.3.4.1	Rekurzivní sestup . . . . .	57
3.3.4.2	Nerekurzivní prediktivní přístup . . . . .	57
<b>4</b>	<b>Analýza řešení</b>	<b>60</b>
4.1	Případy užití . . . . .	61
4.1.1	Obecné případy užití . . . . .	61



4.1.2	Případ užití - UC0 . . . . .	62
4.2	Požadavky . . . . .	63
4.2.1	Funkční požadavky . . . . .	63
4.2.2	Nefunkční požadavky . . . . .	65
<b>5</b>	<b>Návrh uživatelského rozhraní</b>	<b>66</b>
5.1	Flow mapa . . . . .	66
5.2	Hlavní navigace . . . . .	66
5.3	Obrazovka - Login . . . . .	67
5.4	Obrazovka - Today . . . . .	67
5.5	Obrazovka - Game . . . . .	68
5.6	Obrazovka - Playground . . . . .	70
5.6.1	Obrazovka - Tools . . . . .	71
5.6.2	Obrazovka - Actions . . . . .	72
5.6.3	Obrazovka - Step by Step . . . . .	73
5.7	Obrazovka - Statistics . . . . .	74
5.8	Obrazovka - Library . . . . .	75
5.9	Obrazovka - Settings . . . . .	75
<b>6</b>	<b>Návrh architektury</b>	<b>76</b>
6.1	Diagram komponent . . . . .	76
6.2	iOS . . . . .	78
6.2.1	MVC . . . . .	78
6.2.2	MVVM . . . . .	79
6.2.3	Dynamický framework . . . . .	79
6.3	Doménový model . . . . .	81
6.3.1	iOS . . . . .	81
6.3.2	Server . . . . .	81
<b>7</b>	<b>Implementace</b>	<b>83</b>
7.1	Nástroje . . . . .	83
7.1.1	Xcode . . . . .	83
7.1.2	Carthage . . . . .	83
7.1.3	Swiftgen . . . . .	83
7.1.4	PyCharm . . . . .	83
7.1.5	GIT . . . . .	84
7.2	Knihovny . . . . .	84
7.2.1	RxSwift, RxCocoa . . . . .	84
7.2.2	Alamofire . . . . .	84
7.2.3	Charts . . . . .	84
7.2.4	CoreData . . . . .	84
7.2.5	Metal, SpriteKit . . . . .	85

7.2.6	Crashlytics . . . . .	85
7.2.7	Django . . . . .	85
7.3	Prvky architektury . . . . .	85
7.3.1	Observer . . . . .	85
7.3.2	Composite . . . . .	86
7.3.3	Adapter . . . . .	87
7.3.4	Facade . . . . .	88
7.3.5	Iterator . . . . .	89
7.3.6	Strategy . . . . .	90
7.3.7	Factory Method . . . . .	91
7.3.8	Singleton . . . . .	92
<b>8</b>	<b>Ukázka hotového řešení</b>	<b>93</b>
<b>9</b>	<b>Závěr</b>	<b>94</b>
<b>10</b>	<b>Přílohy</b>	<b>101</b>
10.1	Příloha A . . . . .	101
10.1.1	Příloha v IS EDISON . . . . .	101

## Seznam použitých symbolů a zkratk

**UI** User Interface

**UX** User Experience design

**GPGPU** General Purpose Graphics Processing Unit

**UC** Use Case

**REST** Representational State Transfer

**API** Application Programming Interface

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**SDK** Software Development Kit

**IDE** Integrated Development Environment

**UML** Unified Modeling Language

**LL** Left to right, Leftmost derivation

**LR** Left to right, Rightmost derivation

**JSON** JavaScript Object Notation

**MVC** Model - View - Controller

**MVVM** Model - View - ViewModel

**iOS** Název mobilního operačního systému od společnosti Apple Inc.

**macOS** Název desktopového operačního systému od společnosti Apple Inc.

**LU** Lower - Upper rozklad matice, resp. dolní trojúhelníková a horní trojúhelníková matice

## Seznam ilustrací a seznam tabulek

### Seznam ilustrací

1	Chomského hierarchie [27] . . . . .	25
2	Hierarchie výpočetních modelů [27] . . . . .	26
3	Ukázka konkrétního nedeterministického konečného automatu . . . . .	27
4	Levý derivační syntaktický strom . . . . .	30
5	Pravý derivační syntaktický strom . . . . .	30
6	Ukázka konkrétního neorientovaného grafu . . . . .	35
7	Jednoduchý konečný automat pro prázdné slovo $\epsilon$ . . . . .	40
8	Jednoduchý konečný automat pro symbol $a$ . . . . .	40
9	Konečný automat pro zřetězení dvou automatů . . . . .	40
10	Konečný automat pro sjednocení dvou automatů . . . . .	41
11	Konečný automat pro iteraci automatu . . . . .	41
12	Kroky překladače . . . . .	53
13	Ukázka převodu výrazu na syntaktický strom . . . . .	54
14	Struktura nerekurzivního prediktivního parseru . . . . .	59
15	Distribuční systém - základní komponenty . . . . .	60

16	Obecné případy užití iOS aplikace . . . . .	61
17	Návrh - Flow mapa iOS . . . . .	67
18	Obrazovka - Login . . . . .	68
19	Obrazovka - Today . . . . .	68
20	Obrazovka - Game . . . . .	70
21	Obrazovka - Game, hraní . . . . .	70
22	Obrazovka - Playground . . . . .	72
23	Obrazovka - Tools . . . . .	72
24	Obrazovka - Actions . . . . .	73
25	Obrazovka - Step by Step . . . . .	73
26	Obrazovka - Statistics . . . . .	74
27	Obrazovka - Statistics, detail . . . . .	74
28	Obrazovka - Library . . . . .	75
29	Obrazovka - Settings . . . . .	75
30	Diagram komponent . . . . .	77
31	MVC architektura . . . . .	78
32	MVVM architektura . . . . .	79
33	Statický framework [9] . . . . .	80
34	Dynamický framework [9] . . . . .	80
35	iOS databázové relační schéma . . . . .	82
36	Serverové databázové relační schéma . . . . .	82
37	Obecný návrhový vzor observer . . . . .	86
38	Konkretní návrhový vzor observer . . . . .	86
39	Obecný návrhový vzor composite . . . . .	87
40	Konkretní návrhový vzor composite . . . . .	87
41	Obecný návrhový vzor adapter . . . . .	88
42	Konkretní návrhový vzor adapter . . . . .	88
43	Obecný návrhový vzor facade . . . . .	89
44	Konkretní návrhový vzor facade . . . . .	89
45	Obecný návrhový vzor iterator . . . . .	90
46	Konkretní návrhový vzor iterator . . . . .	90
47	Obecný návrhový vzor strategy . . . . .	91
48	Konkretní návrhový vzor strategy . . . . .	91
49	Obecný návrhový vzor factory method . . . . .	91
50	Konkretní návrhový vzor factory method . . . . .	92
51	Obecný návrhový vzor singleton . . . . .	92
52	Ukázka - konečný automat . . . . .	93
53	Ukázka - matice . . . . .	93
54	Ukázka - genetický algoritmus . . . . .	93
55	Ukázka - bezkontextové gramatiky . . . . .	93

## Seznam tabulek

1	Shrnutí existujících řešení . . . . .	21
2	UC0 - Pracovní plocha - popis . . . . .	62
3	Případ užití UC0 - Pracovní plocha . . . . .	63

## Seznam rovnic a vzorců

1	Ukázka prvku matice . . . . .	23
2	Diagonální matice . . . . .	23
3	Jednotková matice . . . . .	24
4	Transponovaná matice . . . . .	24
5	Násobení matic . . . . .	24
6	Minimalizace rozdílu součtu čtverců shluků . . . . .	31
7	Eukleidovská metrika . . . . .	32
8	Manhattanská metrika . . . . .	33
9	Čebyševova metrika . . . . .	33
10	Graf . . . . .	36
11	FIRST funkce . . . . .	55
12	FOLLOW funkce . . . . .	55
13	LL(1) - první pravidlo . . . . .	56
14	LL(1) - druhé pravidlo . . . . .	57
15	Rozkladová tabulka . . . . .	58

## Seznam algoritmů

1	Gaussova eliminační metoda . . . . .	38
2	Gauss-Jordánová eliminační metoda . . . . .	39
3	Převod NFA na DFA . . . . .	43
4	Eliminace nedosažitelných stavů . . . . .	44
5	Sjednocení ekvivalentních stavů (Hopcroft) . . . . .	45
6	Simulace konečného automatu . . . . .	46
7	Odstranění neterminálů, které negenerují terminální slovo . . . . .	47
8	Odstranění nedosažitelných neterminálů . . . . .	48
9	K-means . . . . .	48
10	Genetický algoritmus - Mutace . . . . .	49
11	Genetický algoritmus - Hlavní tok . . . . .	49
12	Genetický algoritmus - Křížení . . . . .	50
13	Force-directed graph drawing . . . . .	51
14	Grafová animace . . . . .	52

# 1 Úvod

V průběhu studia všech vědních oborů, kterými student prochází se setká se spoustou matematických a vědeckých oblastí. Velmi často závisí na jednotlivých vyučujících, jak dobře student pochopí probíranou problematiku. V dnešní době, díky internetu, má student možnost najít materiály právě zde, v případě, že nepochopí vše z výkladu vyučujících. Problém nastává v momentě, kdy si student chce ověřit své znalosti a zjistit, jestli jeho prováděné výpočty jsou správně. Existující software má dva hlavní problémy. První z nich je cena softwaru, kde většinou kvalitní software v tomto směru je velmi drahý. Druhým problémem je samotné použití daného software. Uživatel musí buď používat velmi složité uživatelské rozhraní a nebo speciální programovací jazyk. Z těchto důvodů je pro studenta skoro nemožné použít tento software, když se například učí na zkoušku a potřebuje si během chvilky ověřit své znalosti.

V dnešní době, kde každý má mobilní počítač v kapse, by byla škoda nevyužít tuto možnost právě k výuce a matematickým výpočtům. Toto vedlo k myšlence k vytvoření nového systému, který by běžel na mobilních platformách a umožňoval by nejenom studentům rozšířit své znalosti ve vědeckých oborech s možností provádět vlastní matematické výpočty, operace a vidět postup řešení. Kromě toho by se zde měl nacházet mechanismus na obnovování již naučených dovedností a znalostí, jelikož znalosti, které nejsou pravidelně používány, tak bohužel postupem času jsou velmi rychle zapomínány.

Celá práce je rozdělena do čtyřech hlavních částí. První část se zabývá analýzou současného stavu řešení, jejich porovnání a nalezení nedostatků. Další část se týká návrhu nového řešení, sběrem požadavků a návrhem prototypu uživatelského rozhraní. Třetí část popisuje výpočetní jádro, které systém používá pro všechny výpočty, operace a ostatní algoritmy. Nachází se zde teoretický popis jednotlivých oblastí algebry a teoretické informatiky. Teoretické znalosti jsou doprovázeny popisem použitých algoritmů, které jsou potřeba pro implementaci. Poslední oblast v této části slouží k rozebrání syntaktické analýzy, která slouží ke kontrolování uživatelských vstupů. Poslední část je samotná implementace celého systému. Zde je rozebrána architektura, prvky architektury, použité nástroje a ostatní doplňující implementační prvky.

Celý systém je složen ze třech hlavních částí [2]. První část je iOS mobilní aplikace, se kterou uživatel interaguje [51]. Druhá část je implementace knihovny, která slouží jako výpočetní jádro. Poslední část je serverový backend, který zajišťuje správu uživatelů a uchování uživatelských dat.

Práce přináší i nové poznatky z oblasti uživatelského rozhraní. Ať už vlastní návrh zadávání uživatelských vstupů, nebo například vytvoření speciální animace, založena na teorii grafů. Základní princip animace je takový, že vrcholy se pohybují po viditelné ploše různými směry a rychlosti. Podle vzdálenosti mezi sebou se různě zobrazují hrany mezi jednotlivými vrcholy.

Správné pochopení matematiky a vybraných partií informatiky je základ pro úspěšné pochopení

a práci v oblasti analýzy dat [56]. Z důvodů rozšíření obzorů, znalostí a dovedností v této oblasti (analýza dat) je zvolen postup v implementační části takový, že všechny algoritmy, které jsou použité pro výpočty, operace, syntaktickou analýzu, vykreslování a ostatní podpůrné algoritmy, jsou implementovány ručně, bez použití jakékoliv knihovny. [31, 15]





## 2 Analýza současného stavu řešení

Analýzu současného stavu řešení lze rozdělit podle platformy, pro kterou je software vytvořený. Jedná se o následující hlavní platformy: mobilní, desktopové a webové. Jelikož se práce zaměřuje na vytvoření nového systému pro mobilní platformu iOS, tak zde bude tato část rozepsána detailněji a zbylé dvě (desktopové a webové) budou pouze krátce shrnuty.

### 2.1 Apple iOS

Na platformě iOS se nachází spousta aplikací zaměřených na matematiku, proto zde budou rozebrány pouze ty nejpopulárnější. Analýze již existujících řešení se bude hlavně zabývat tím, co daná aplikace uživatelům nabízí za odvětví resp. disciplíny matematiky, jestli nabízí materiály ke studiu dané problematiky, ověření znalostí uživatelů (například přes předpřipravené příklady, nebo automaticky generované), vysvětlení postupu řešení, či jiné schopnosti.

#### 2.1.1 Photomath

První aplikace, na kterou lze téměř hned narazit, je Photomath [36]. Hlavní a jediné zaměření je práce s rovnicemi. Aplikace je schopná provádět základní úpravy rovnic. Zobrazuje postup řešení. Taktéž je schopná vyřešit soustavu rovnic. Hlavní výhodou Photomath, je možnost zadat vstup pouze vyfocením rovnice na papíře. Následně dojde k rozpoznání a převedení rovnice a uživatel má možnost upravit rovnici, pokud došlo ke špatnému rozpoznání. Druhá volba zadání vstupu je klasickým způsobem přes klávesnici na obrazovce. Je zde taky možnost nechat si vykreslit graf zadané funkce.

Photomath se tedy zabývá pouze řešením rovnic a zobrazením postupu řešení, mimo to nenabízí uživateli nic dalšího. Jako velké plus zde lze určitě zmínit možnost zadání vstupu přes fotoaparát.

#### 2.1.2 Math 42

Další populární aplikace je Math 42 [17]. Opět je zde zaměření na práci s rovnicemi, zobrazení postupu řešení a vykreslení grafu funkce. Nicméně schopnosti aplikace jsou rozšířené o možnosti výpočtů derivací a integrálů. Jsou zde obsaženy velmi jednoduché učební materiály, které pokrývají více oblastí, než je aplikace schopná řešit. Dalším pozitivním plusem je zde možnost využít trénovací a testovací nástroje, které ověří, jestli uživatel dané problematice rozumí. Data pro trénování a testování jsou vždy generovány náhodně, takže uživatel má možnost zkoušet příklady tak dlouho, dokud potřebuje. I přes spoustu funkcionality, které aplikace nabízí má jednu velkou nevýhodu, a to je UI a UX. Z uživatelského hlediska je ovládání velmi špatné i pro základní operace.

Math 42 nabízí spoustu dobrých vlastností oproti Photomath, jako jsou učební materiály, testovací a trénovací nástroje pro ověření znalostí z dané problematiky a rozšíření na matematickou

analýzu (derivace a integrály). Jediná nevýhoda je velmi špatné UI a UX.

### 2.1.3 iMathematics

Zde se objevuje otázka, jestli sem iMathematics zařadit [5]. Aplikace není schopna vyřešit žádný problém, ale pouze otevírá webový prohlížeč s odkazem na WolframAlpha. Důvod proč sem iMathematics byla zařazena je ten, že má dobře propracovaný režim na testování znalostí z algebry. Pro každou disciplínu, například: aritmetiku, geometrii, trigonometrii, analýzu, atd. je zde obsažena sada otázek, resp. problémů. Jejich obtížnost se postupně zvyšuje. Aplikace měří a vyhodnocuje výsledky daného uživatele. Podle výsledků zobrazuje statistiky pro každou disciplínu zvlášť. Uživatel tam má jasný přehled o tom co zvládá a naopak v čem zaostává.

iMathematics kromě testovacího modelu nic jiného nenabízí.

### 2.1.4 FX Math Solver

FX Math Solver se zaměřuje hlavně na prezentaci a ukázky postupu řešení problémů [23]. Jako předchozí aplikace dokáže pracovat se základní algebrou, ale přidává navíc derivace, integrály, matice a nerovnice. Bohužel, co se týká matic, tak dokáže pracovat se vstupem pouze  $2 \times 2$  a  $3 \times 3$ . I přes to, že to je velmi omezující, tak pro studium operací s maticemi a na prezentaci postupu řešení to je dostačující. Nicméně ztrácí to univerzálnost.

Teď zpět k tomu, jak FX Math Solver pracuje s prezentací postupu řešení problémů. Z pohledu uživatele to probíhá skoro jako video. Na obrazovce se postupně zobrazují jednotlivé kroky za sebou a jsou zvýrazněné úpravy, které v současném kroku nastaly. Uživatel se může vrátit zpět a nebo rychleji přejít dopředu. Celý výsledek působí tak, jako kdyby to někdo psal na tabuli. Bohužel i zde je problém s UI a UX. Zadávání vstupů jednotlivých problémů je velmi kostrbaté a probíhá přes klávesnici.

FX Math Solver má skvěle zpracovanou prezentaci postupů řešení uživateli. Dokáže zpracovat a zobrazit postupy pro hodně disciplín algebry.

### 2.1.5 Symbolab Calculator

Symbolab Calculator [49] je na první pohled velmi podobná již zmíněné první aplikaci Photomath. Nicméně zdání klame a aplikace toho nabízí opravdu hodně. Kromě základních rovnic a výrazů, které uživatel může zadat přes fotoaparát, se zde nachází i další disciplíny, například: nerovnice, limity, derivace, integrály, trigonometrie, matice a statistika. Jak lze vidět, tak výběr je opravdu široký. Pro každý problém je zobrazen taktéž postup řešení, krok za krokem.

Bohužel i zde je stejný problém s UI a UX ohledně zadávání vstupů pro daný problém. Autor dokonce pro všechny disciplíny, které aplikace pokrývá, použil jednu klávesnici pro zadávání vstupů, takže se zde nacházejí veškeré symboly, čísla, operátory a další znaky. Úprava již zadaného problémů je opět velmi kostrbatá.

Symbolab Calculator je skvělý software, pokrývající velkou část matematiky. Nabízí zobrazení postupu řešení. Když se přehlídne špatné UI a UX pro zadání vstupu, tak je to jedna z nejlepších aplikací.

### 2.1.6 Mathway

Na první pohled aplikace Mathway [32] zaujme svým jiným přístupem ohledně UI. Jako ostatní zmíněné aplikace i tato je schopna zpracovat více věcí od algebry přes integrály, matice, statistiku až po chemii. Jak bylo zmíněno na začátku, přístup aplikace s jakým se s ním pracuje, funguje na principu klasického chatu, tak jak ho lze znát ze sociálních sítí. Uživatel vloží vstup a odešle se zpráva se vstupem a požadavkem co se má s daným vstupem provést za operaci. Výsledek je vrácen uživateli jako zpráva od systému, kde je vidět výsledek řešení. Uživatel má dále možnost si zobrazit postup řešení, přehledně krok za krokem. Nachází se zde taky klasická klávesnice se symboly pro zadání vstupů. Nicméně ovládání je mnohem lépe vyřešeno. Aplikace se dokonce snaží odhadnout co chcete vložit, resp. napsat za vstup a nabízí uživateli na výběr z různých vstupů, na které stačí kliknout a není potřeba ručně dopsat vstup. Je to stejný princip jak když se píše text v libovolném jazyce a systém se snaží odhadnout jaké slovo uživatel chce napsat a tím tak urychlit psaní textu, resp. v tomto případě zadání problémů.

Mathway pokrývá velkou část matematiky a i jiné vědecké disciplíny. Zaujme především nekonvenčním zpracováním UI.

### 2.1.7 Khan Academy

Khan Academy [29] není potřeba snad ani představovat. Pro člověka, který se zajímá o libovolnou vědní disciplínu, je tohle velmi známý projekt. Jedná se o projekt, který pokrývá veškeré vědecké disciplíny od matematiky přes fyziku, chemii, teoretickou informatiku, biologii a nespočetně mnoho dalšího. Software obsahuje audiovizuální učební materiály, stejně tak i textové. Dále se zde nachází propracovaný systém trénování a cvičení probíraných problematik. Uživatel získává za dokončené kurzy body, které se globálně sčítají. V průběhu cvičení uživatel za získané body získává různé znaky za dosavadní úspěchy a zvedá se mu úroveň. Cvičící model uživatele motivuje se dále vzdělávat a zlepšovat své znalosti. Nicméně i přes perfektní materiály a cvičící model aplikace neobsahuje možnost zadat vlastní vstup a vyřešit ho. Slouží pouze ke studiu. Projekt začal před pár lety jeden člověk a v současné době je zde zapojeno více než 150 lidí. To co určitě stojí za zmínku, tak celý projekt je kompletně zdarma pro všechny.

Khan Academy je perfektní nástroj pro vzdělávání, kde uživatel může získat informace z vědních disciplín i mimo akademické prostředí. Drobným nedostatkem zde může být nemožnost zadávání vlastních vstupů a vidět postup řešení.

### 2.1.8 Brilliant

Naprosto totožný projekt jako předchozí Khan Academy. Brilliant [14] se liší hlavně v tom, že je zpoplatněný. Nachází se zde velmi propracované textové a vizuální materiály. Pokrytí vědeckých disciplín je přibližně podobné jako u Khan Academy. Taktéž se zde nachází propracovaný trénovací model. Mimo to jsou obsaženy týdenní a denní problémy, které uživatelé mohou vyřešit a navzájem se porovnávat. Projekt obsahuje pouze vzdělávací materiály a trénovací model, není zde už možnost zadat vlastní vstup a vidět postup řešení.

Brilliant je opět perfektní nástroj pro vzdělávání. Oproti Khan Academy je nevýhoda v ceně, která je ve stovkách Kč za měsíc.

## 2.2 Desktop a Web

Desktopové programy se sem úplně nehodí, co se týká porovnání vůči aplikacím z platformy iOS. Jelikož tyto programy jsou masivní, desítky let vyvíjeny a obsahují snad vše co lze nelézt v matematice a ostatních vědních oborech. Jsou zde zařazeny pouze z informativního důvodu. Desktopové programy mají často dodělané webové rozhraní, které lze použít on-line a není tak potřeba instalovat žádný software do svého počítače.

### 2.2.1 Wolfram Mathematica a Wolfram Alpha

Asi nejznámější masivní a robustní software pro matematiku a ostatní vědní disciplíny je Mathematica [54]. Software používá vlastní programovací jazyk, kterým je možné zadat vstup. Druhý způsob zadání vstupu je přes UI, takže uživatel vždy není nucen se učit jejich specifický jazyk. K desktopovému programu Mathematica bylo vytvořeno webové rozhraní Wolfram Alpha [55]. Díky tomu není potřeba instalovat žádný software. Wolfram Alpha používá na pozadí jádro z Mathematica, takže i zde je možnost použít ten stejný programovací jazyk pro zadání složitějších vstupů.

Jelikož se jedná o masivní výpočetní software, který je vyvíjený desítky let, tak tomu odpovídá i cena, která je v tisících Kč za rok pro studenty a domácí použití.

### 2.2.2 Matlab

Matlab [50] je software velmi podobný Mathematica. Obsahuje taktéž veškeré vědní disciplíny a k ním výpočetní jádro. Liší se hlavně v tom, že Matlab je více zaměřený na data a práci s nimi, jako například: analýza dat, neuronové sítě, zpracování signálu atd.

Matlab je taktéž zpoplatněný. Nicméně, cena se pro studenty nebo domácí použití šplhá až na desítky tisíc Kč za rok.

## 2.3 Shrnutí

Z analýzy současného stavu řešení pro platformu iOS vyplývá, že se veškerý existující software zaměřuje na algebru a vždy na její stejné části (například: rovnice, výrazy, derivace, integrály).

Několik aplikací nabízí vlastní učební materiály ke studiu a to jak v textové tak i v audiovizuální podobě. Velmi podobně na tom je i trénovací a testovací model, který pomáhá uživateli dosahovat větších znalostí a zlepšovat se v různých disciplínách matematiky.

Jak jde vidět, tak na iOS chybí software, který by se zaměřil na části matematiky nebo jiných vědních disciplín, které existující řešení nepokrývají. Například z algebry práce s maticemi. Mimo jiné nelze nalézt software pro práci s výrokovou logikou, teoretickou informatikou, matematikou pro ekonomii, nebo abstraktní algebrou.

Software	UI a UX	Postup	Materiály	Model	Zaměření	Vstup	Cena
Photomath	Dobré	ANO	NE	NE	Rovnice	ANO	Z
Math 42	Velmi špatné	ANO	ANO	ANO	Algebra	ANO	Z
iMathematics	Dobré	NE	ANO	ANO	Algebra	NE	ZP
FX Math Solver	Špatné	ANO	NE	NE	Algebra	ANO	ZP
Symbolab Calculator	Velmi špatné	ANO	ANO	NE	Algebra	ANO	Z
Mathway	Dobré	ANO	NE	NE	Algebra	ANO	Z
Khan Academy	Dobré	NE	ANO	ANO	Vše	NE	Z
Brilliant	Dobré	NE	ANO	ANO	Vše	NE	P

Tabulka 1: Shrnutí existujících řešení

#### Vysvětlivky k tabulce:

Sloupec *UI a UX* - zaznamenává, kvalitu uživatelského rozhraní a kvalitu zadávání vstupů.

Sloupec *Postup* - zda aplikace zobrazuje postup řešení.

Sloupec *Materiály* - zda aplikace poskytuje vlastní učební materiály.

Sloupec *Model* - zda aplikace poskytuje trénovací a testovací model, který uživatelé posouvá dále.

Sloupec *Zaměření* - Hlavní zaměření aplikace

Sloupec *Vstup* - Možnost zadat vlastní vstup, který aplikace vyřeší.

Sloupec *Cena* obsahuje tři hodnoty: **P**, **Z**, **ZP**, kde:

**Placené** - znamená, že je služba pouze placená

**Zdarma** - znamená, že je celá služba zdarma

**ZP** (Zdarma & Placené) - znamená, že služba je zdarma s omezením

### 3 Návrh výpočetního modelu

Návrh výpočetního modelu nebo jádra slouží k definování toho, co aplikace bude schopna zpracovávat a jak to bude dělat. Budou zde rozebrány definice, pseudokódy, funkce (vlastnosti aplikace) a ostatní podpůrné prostředky pro správný běh výsledného systému. Jedná se o nejdůležitější a nejnáročnější část celého systému.

Z analýzy současného stavu řešení vyplynulo, že většina existujícího software se zaměřuje na algebru a její partie, jako například: rovnice, výrazy, derivace nebo integrály. Z toho důvodu se tento nově vytvářený systém bude ze začátku zabývat tím, co v současné době chybí na platformě iOS a oblastí z vlastních požadavků na výsledný systém. Proto hlavní zaměření bude na teoretickou informatiku, částečně na algebru a pak vybrané algoritmy ze strojového učení a evolučních algoritmů. Systém v první fázi bude schopen zpracovat, respektive pokrývat, následující oblasti:

1. Matice.
2. Regulární jazyky.
3. Konečné automaty.
4. Bezkontextové gramatiky.
5. Shlukový algoritmus - K-means.
6. Evoluční algoritmus - Genetický algoritmus.

K výše uvedenému seznamu lze ještě doplnit jednu hru, která byla implementována čistě ze zajímavého (vizuálního) hlediska. Jedná se o hru s názvem Game of Life, která je založena na buněčných automatech, určena pro nula hráčů. Všechny vypsane oblasti budou v následujících kapitolách rozebrány. Nejprve bude popsána stručná teorie, která definuje základní teoretické znalosti. Potom na tom bude stavět kapitola, ve které budou rozepsány nejzajímavější jednotlivé funkce, které bude systém schopen zpracovat.

#### 3.1 Teoretický základ

Kapitola se zabývá stručným popisem a definicí výše vypsanych oblastí a matematických partií. Budou popsány a vysvětleny základní definice a ostatní dodatečné definice potřebné v dalším textu. Jelikož se nejedná o práci zaměřenou na popis matematických struktur, definic a důkazů, tak definice budou velmi stručné. Předpokládá se, že čtenář už má základní znalosti z popisovaných oblastí a partií.

##### 3.1.1 Matice

Matice mají rozsáhlé využití v algebře a jde s nimi provádět spoustu operací. Jedna ze základních aplikací je řešení soustav rovnic pomocí matic [20].



### 3.1.1.1 Definice

Nechť  $A$  je komutativní těleso (respektive algebraická struktura pole) a  $n, m$  jsou pozitivní celé čísla. Množina  $n \times m$  matic  $A^{n \times m}$  je definována jako  $A$ -modul s prvky  $a_{i,j}$ , kde  $1 \leq i \leq n$  a  $1 \leq j \leq m$ . Prvek  $a_{i,j}$  představuje matici se všemi hodnotami rovny 0, kromě hodnoty na pozici  $i$ -tého řádku a  $j$ -tého sloupce, která se rovná 1. Všechny ostatní matice jsou lineární kombinací  $a_{i,j}$ , kde koeficienty určují hodnoty matice [20]. Příklad pro matici  $3 \times 4$  a prvek  $a_{2,3}$ :

$$a_{2,3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

A teď jednodušeji řečeno bez korektní matematické definice. Matice je obdélníkové pole hodnot, uspořádané do řádku a sloupců.

### 3.1.1.2 Speciální typy

Při určitých okolnostech a vlastnostech matice mohou nastat speciální případy. Specifických typů matic je velmi mnoho, takže zde budou vypsány pouze základní typy.

#### Čtvercová matice

Je matice, která má stejný počet řádků a sloupců. Neboli kde  $n = m$ . Jestliže matice není čtvercová, tak se označuje jako obdélníková [22].

#### Nulová matice

Nulová matice je taková matice, která má všechny hodnoty nulové [22]. Neboli  $\forall i \in \{1, 2, \dots, n\}, \forall j \in \{1, 2, \dots, m\} : a_{i,j} = 0$ .

#### Diagonální matice

Je to matice, která má všechny hodnoty nulové, kromě hlavní diagonály. Na hlavní diagonále mohou, ale nemusí být nuly [22]. Neboli  $\forall i \in \{1, 2, \dots, n\}, \forall j \in \{1, 2, \dots, m\}$  platí, že:

$$A_{i,j} = \begin{cases} x, & i = j \\ 0, & i \neq j \end{cases} \quad (2)$$

#### Jednotková matice

Za jednotkovou matici, lze považovat pouze takovou matici, která má hodnoty na hlavní diagonále pouze 1 a všechny ostatní hodnoty 0 [22]. Neboli  $\forall i \in \{1, 2, \dots, n\}, \forall j \in \{1, 2, \dots, m\}$  platí, že:

$$A_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad (3)$$

### Schodovitá matice

Je matice, kde vedoucí prvky na řádce vizuálně připomínají schody. Matice má nulové řádky na konci (nebo žádné nulové řádky) a každý řádek má více nul na začátku, než jeho předchozí řádek [22].

### Transponovaná matice

Transponovaná matice je taková matice, která má prohozené řádky za sloupce [22]. Neboli k matici  $A$  je matice  $A^T$ , pro kterou platí:

$$a_{i,j} = A_{j,i}^T \quad (4)$$

#### 3.1.1.3 Operace

Pro práci s maticemi jsou definované základní operace: součet, rozdíl, násobení skalárem a násobení matice maticí. První tři operace jsou přímočaré. Nicméně pro využití operace součet a rozdíl musí matice splňovat podmínku, že obě dvě matice mají stejný rozměr. Jinak řečeno, mají stejný počet řádků a sloupců. Vzhledem k tomu, že první tři jmenované operace jsou jednoduché bude blíže vydefinována jen operace násobení matic [22].

### Násobení matic

Násobení matic už tak přímočaré není. Jednoduše řečeno, násobí se řádek z první matice se sloupcem z druhé matice a výsledek se vloží na příslušnou pozici ve výsledné matici. Musí být dodržena podmínka, že počet sloupců první matice je roven počtu řádků druhé matice [22]. Neboli nechť je matice  $A$  o rozměrech  $n \times m$  a matice  $B$  o rozměrech  $m \times p$ , potom pro  $\forall i \in \{1, 2, \dots, n\}, \forall j \in \{1, 2, \dots, p\}$  a operaci součin matic  $A \cdot B$  platí, že:

$$(A \cdot B)_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j} = a_{i,1} \cdot b_{1,j} + a_{i,2} \cdot b_{2,j} + \dots + a_{i,m} \cdot b_{m,j} \quad (5)$$

#### 3.1.1.4 Základní úpravy

Kromě základních operací, které můžeme provádět s maticemi, máme ještě základní úpravy, které aplikujeme pouze na jednu matici. Tyto úpravy se nazývají základní elementární úpravy [22]. Po aplikování elementární úpravy na matici  $A$  vznikne nová matice  $A'$ .

Základní řádkové úpravy jsou následující:

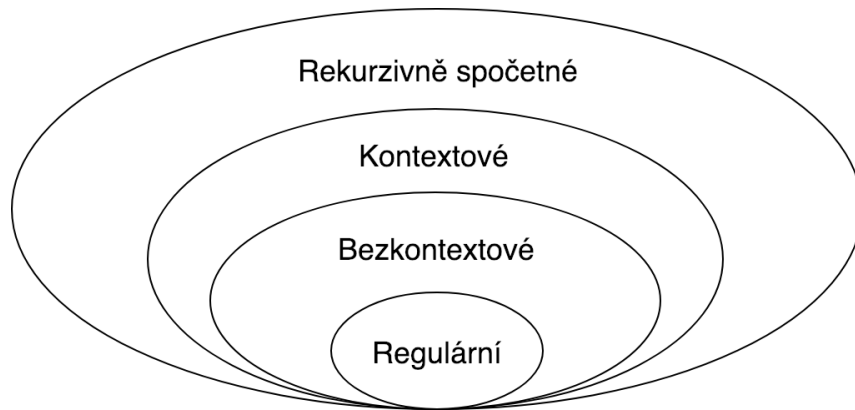
1. Přičtení násobku jednoho řádku k druhému.
2. Vynásobení řádku nenulovým prvkem.

3. Výměna dvou řádků mezi sebou.

Pro každou provedenou elementární úpravu nad maticí  $A$ , existuje inverzní elementární úprava, která převede matici  $A'$  zpět na původní matici  $A$  [22].

### 3.1.2 Regulární jazyky

Regulární jazyky mohou být definované více způsoby. Regulární jazyky jsou nejzákladnější formální jazyky v Chomské hierarchii (Obrázek 1). Chomského hierarchie popisuje skupiny formálních gramatik, které generují formální jazyky [27].



Obrázek 1: Chomského hierarchie [27]

#### 3.1.2.1 Definice

Regulární jazyky nad abecedou  $\Sigma$  jsou definovány jako [27]:

1. Prázdný jazyk  $\%_0$  a prázdný jazyk obsahující prázdný řetězec  $\{\epsilon\}$  jsou regulární jazyky.
2.  $\forall a \in \Sigma$  platí, že jazyk  $\{a\}$  je regulární jazyk.
3. Necht  $A$  a  $B$  jsou regulární jazyky, potom  $A \cup B$  (sjednocení),  $A \cdot B$  (konkatenace) a  $A^*$  (iterace) jsou regulární jazyky.
4. Žádné další jazyky nad abecedou  $\Sigma$  nejsou regulární.

#### Zajímavé vlastnosti regulárních jazyků

Regulární jazyk:

1. Je akceptován nedeterministickým konečným automatem.
2. Je akceptován deterministickým konečným automatem.
3. Může být vygenerován regulární gramatikou.
4. Může být vygenerován prefixovou gramatikou.
5. Může být akceptován Turingovým strojem.

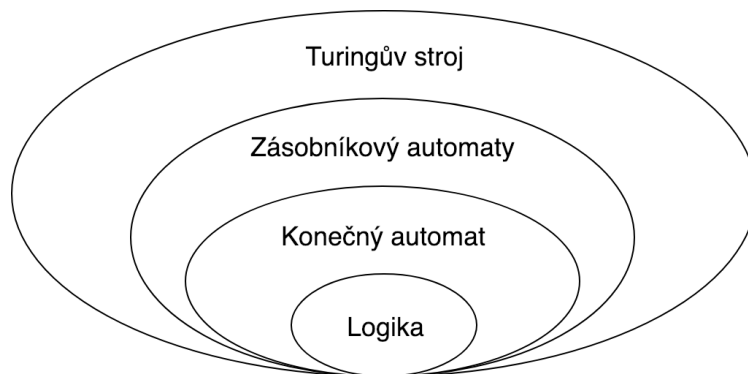
### 3.1.2.2 Operace

Operací nad jazyky je definovaných několik. Zde následují základní z nich. Necht  $L$  a  $M$  jsou regulární jazyky, potom [27]:

1. Sjednocení,  $L \cup M = \{w | w \in L \vee w \in M\}$ .
2. Průnik,  $L \cap M = \{w | w \in L \wedge w \in M\}$ .
3. Rozdíl,  $L - M = \{w | w \in L \wedge w \notin M\}$ .
4. Zřetězení,  $L \cdot M = \{uv | u \in L \wedge v \in M\}$ .
5. Iterace (obecná),  $L^* = L^0 \cup L^1 \cup L^2 \dots = \bigcup_{i=0}^{\infty} L^i$ .
6. Otočení (zrcadlení),  $L^R = \{u^R | u \in L\}$ .

### 3.1.3 Konečné automaty

Konečný automat, nebo stavový stroj, je matematický teoretický výpočetní model (Obrázek 2). Jedná se o abstraktní stroj, který se může nacházet v jednom z možných stavů. Přejít z jednoho stavu do jiného se provádí na základě reakce na vstup. Automat začíná z počátečního stavu a končí při dočtení celého vstupu. Při skončení mohou nastat dvě situace. Automat se buď nachází ve stavu, který je v přijímací množině a nebo ne. Rozlišují se dva typy konečných automatů a to: deterministické konečné automaty a nedeterministické konečné automaty. Deterministický konečný automat je schopen rozpoznat regulární jazyk [27].



Obrázek 2: Hierarchie výpočetních modelů [27]

#### 3.1.3.1 Definice - Deterministický konečný automat

Formální definice deterministického konečného automatu  $M$  je uspořádána pětice  $(Q, \Sigma, \delta, q_0, F)$ , kde [27]:

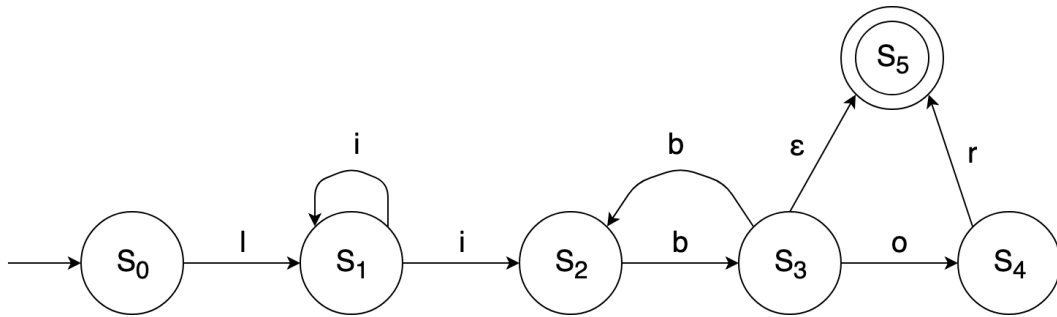
1.  $Q$  - konečná neprázdná množina stavů.
2.  $\Sigma$  - konečná neprázdná množina symbolů, nazývaná abeceda.

3.  $\delta$  - (stavová) přechodová funkce, která popisuje přechod z jednoho stavu do druhého. Neboli  $\delta : Q \times \Sigma \rightarrow Q$ .
4.  $q_0$  - počáteční stav. Neboli  $q_0 \in Q$ .
5.  $F$  - množina přijímacích stavů. Neboli  $F \subseteq Q$ .

### 3.1.3.2 Definice - Nedeterministický konečný automat

Nedeterministický konečný automat je velmi podobný deterministickému konečnému automatu (Obrázek 3). Rozdíl je hlavně v přechodové funkci a to tak, že z jednoho stavu může automat přejít do více stavů najednou přečtením pouze jednoho symbolu ze vstupu. Nedeterministický konečný automat může být ještě dále více zobecněn pomocí epsilon přechodů, označovaný jako  $\epsilon$ -přechod. Na základě  $\epsilon$ -přechodu může automat přejít do jiného stavu i bez přečtení symbolu ze vstupu. Změna ve formální definici je pouze v přechodové funkci [27]:

1.  $\delta$  - (stavová) přechodová funkce, s  $\epsilon$ -přechodem. Neboli  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$ , kde  $P(Q)$  je potenční množina.



Obrázek 3: Ukázka konkrétního nedeterministického konečného automatu

### 3.1.3.3 Zásobníkový automat

Zásobníkový automat rozšiřuje konečný automat o možnost využití zásobníku. Přechod z jednoho stavu do druhého probíhá na základě rozhodování podle aktuálního stavu, symbolu na vrcholu zásobníku a symbolu na vstupu [27].

Formálně je zásobníkový automat  $M$  definován jako uspořádaná sedmice  $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ , kde [27]:

1.  $Q$  - konečná množina stavů.
2.  $\Sigma$  - konečná množina symbolů (abeceda).
3.  $\Gamma$  - konečná množina symbolů zásobníku, nazývaná abeceda zásobníku.
4.  $\delta$  - přechodová funkce. Neboli  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^* \rightarrow P(Q)$ .

5.  $s_0$  - počáteční stav. Neboli  $q_0 \in Q$ .
6.  $Z$  - počáteční symbol na zásobníku. Neboli  $Z \in \Gamma$ .
7.  $F$  - množina přijímacích stavů. Neboli  $F \subseteq Q$ .

### 3.1.4 Bezkontextové gramatiky

Bezkontextová gramatika je speciální typ formální gramatiky. Je to konečná množina přepisovacích pravidel, ve tvaru  $A \rightarrow \alpha$ , kde  $A$  je neterminál symbol a  $\alpha$  je řetězec, který může obsahovat jak terminální, tak neterminální symboly. Z toho vyplývá, že neterminální symboly se musí dále přepsat a neterminální symboly naopak už dále přepsat nejdou. Bezkontextovou gramatiku je schopen rozpoznat zásobníkový automat [27].

#### 3.1.4.1 Definice

Formálně je bezkontextová gramatika  $G$  definována jako uspořádaná čtveřice  $(V, \Sigma, R, S)$ , kde [27]:

1.  $V$  - je konečná množina neterminálních symbolů.
2.  $\Sigma$  - je konečná množina terminálních symbolů, nazývané taky jako abeceda gramatiky. Průnik mezi neterminálních symbolů a terminálních symbolů je prázdný. Neboli  $V \cap \Sigma = \emptyset$ .
3.  $R$  - je konečná množina přepisovacích pravidel. Neboli  $V \rightarrow (V \cup \Sigma)^*$ .
4.  $S$  - je počáteční neterminální symbol. Neboli  $S \in V$ .

#### 3.1.4.2 Derivace

Derivace vstupu (textového řetězce) pro gramatiku, je posloupnost přepisovacích pravidel. Odvození (derivování) řetězce z gramatiky začíná z počátečního neterminálu. Jestli je celý řetězec odvozen gramatikou, respektive jde najít taková posloupnost přepisovacích pravidel, která vygeneruje zadaný textový řetězec na vstupu, tak to znamená, že textový řetězec patří do jazyka gramatiky. Posloupnost přepisovacích pravidel lze znázornit stromem [27].

Při použití (aplikování) řetězce pravé strany přepisovací pravidla, které obsahuje jak terminální tak neterminální symboly, lze pro stejný vstupní řetězec odvodit více různých derivačních stromů. Podle toho, jestli vždy první přepíšeme nejlevější neterminál, nebo naopak nejpravější neterminál, tak se jedná buď o levou derivaci, nebo o pravou derivaci. Pokud gramatika generuje pro vstupní řetězec více derivačních stromů, tak se tato gramatika nazývá nejednoznačná gramatika [27].

### 3.1.4.3 Ukázkový příklad

U gramatik je vhodné názorně předvést výše popsané teoretické definice na příkladu, jelikož z teoretických definic nemusí být hned vše každému jasné.

Nechť  $G$  je bezkontextová gramatika definována přepisovacíma pravidly:

1.  $S \rightarrow S + S$
2.  $S \rightarrow 0$
3.  $S \rightarrow 1$

Tato gramatika je například schopná vygenerovat následující řetězec:  $0 + 0 + 1$ . Derivace řetězce může být následující:

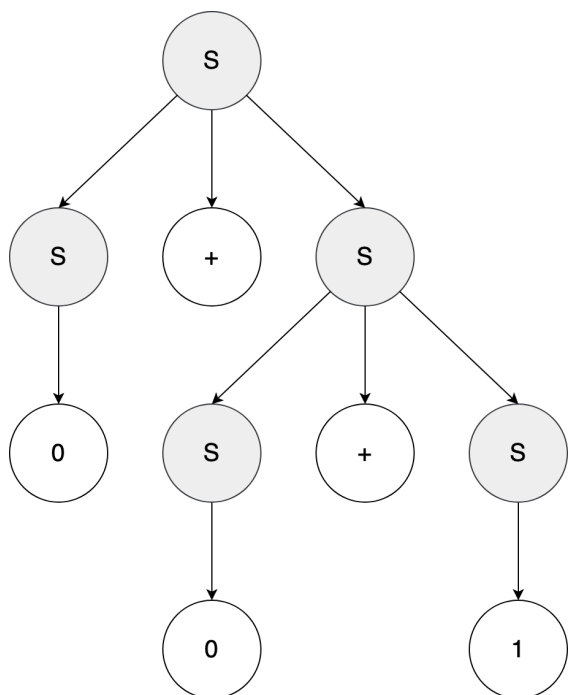
1.  $S \rightarrow S + S$  (první pravidlo)
2.  $S + S \rightarrow 0 + S$  (druhé pravidlo na první  $S$ )
3.  $0 + S \rightarrow 0 + S + S$  (první pravidlo na první  $S$ )
4.  $0 + S + S \rightarrow 0 + 0 + S$  (druhé pravidlo na první  $S$ )
5.  $0 + 0 + S \rightarrow 0 + 0 + 1$  (třetí pravidlo na první  $S$ )

Podle toho, který neterminál bude přepsán jako první, například vždy nejlevější, tak to bude levá derivace (viz. předchozí ukázka), naopak pokud bude přepsán vždy nejpravější neterminál, tak to bude pravá derivace, která vypadá následně pro předchozí řetězec:

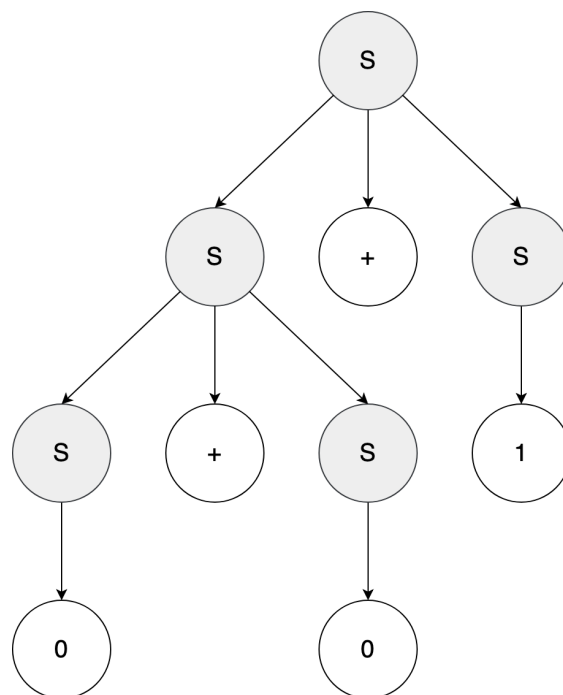
1.  $S \rightarrow S + S$  (první pravidlo)
2.  $S + S \rightarrow S + S + S$  (první pravidlo na druhé  $S$ )
3.  $S + S + S \rightarrow S + S + 1$  (třetí pravidlo na třetí  $S$ )
4.  $S + S + 1 \rightarrow S + 0 + 1$  (druhé pravidlo na druhé  $S$ )
5.  $S + 0 + 1 \rightarrow 0 + 0 + 1$  (druhé pravidlo na první  $S$ )

Derivaci může být znázorněna stromem, který se nazývá abstraktní syntaktický strom. Zde jsou názorné ukázky pro předchozí příklady, jak levou (Obrázek 4), tak pravou derivaci (Obrázek 5).





Obrázek 4: Levý derivační syntaktický strom



Obrázek 5: Pravý derivační syntaktický strom

### 3.1.5 Machine learning

Machine learning, neboli strojové učení, se zabývá algoritmy a statistickými metodami, které může počítač použít k provedení specifické akce, bez nutnosti implicitně implementovat danou akci. V dnešní době, machine learning hraje klíčovou roli v mnoha oblastech. Často se také uvádí, že machine learning je podmnožinou umělé inteligence. Algoritmy fungují na základním principu, kde první probíhá fáze učení, neboli vytvoření modelu a potom následné využití, či vyhodnocení natrénovaného modelu. Pro fázi učení se používá datová sada označovaná jako trénovací množina. Naopak pro vyhodnocení natrénovaného modelu se používá datová sada označovaná jako testovací množina [34].

#### 3.1.5.1 Základní typy učení

Existuje několik metod nebo postupů, kterými se algoritmus učí, respektive vytváří model [34].

1. Učení s učitelem obsahuje trénovací sadu se správnými výsledky, algoritmus zná správný výsledek.
2. Učení bez učitele, algoritmus neví, jaký je správný výsledek.
3. Posílené učení, je učení pomocí zpětné vazby.
4. Kombinace učení s učitelem a bez učitele.

### 3.1.5.2 Základní typy úloh

Algoritmy lze rozdělit taky podle oblasti zaměření [34].

1. Klasifikace - dochází k rozdělení dat do několika kategorií nebo tříd (učení s učitelem).
2. Shlukování - nalezení struktur, vzorů nebo shluků ve stupních datech (učení bez učitele).
3. Regrese - predikce výstupu na základě vstupu (učení s učitelem).

### 3.1.5.3 K-means

K-means slouží ke shlukování dat v  $n$ -dimenzionálním prostoru do  $k$  shluků, kde na konci algoritmu, každá instance (záznam) z datové sady náleží do nějakého  $k$  shluku. Jedná se o jeden z nejzákladnějších algoritmů pro shlukování. Výpočetně se jedná o NP-těžký problém [34].

Pro danou datovou sadu  $(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n)$  obsahující záznamy, jako  $n$ -dimenzionální vektor, dojde k rozdělení datové sady do  $k$  množin  $S = (S_1, S_2, \dots, S_k)$ . Proto musí platit, že  $k \leq n$ . Dochází zde k minimalizaci rozdílů čtverců uvnitř každého shluku (odchylka), respektive rozdíl mezi centroidem shluku (střed shluku) a záznamem, který patří do daného shluku. Toto se počítá pro každý záznam v každém shluku a výsledek je pro každý shluk součet jeho chyby, které můžeme sečíst a mít jedno globální číslo, které udává globální chybu celkového shluknutí dat do  $k$  shluků. Toto číslo se často označuje jako SSE, neboli anglicky Error Sum of Squares [34].

Proto platí, že jestli  $k = n$ , potom  $SSE = 0$ . Jelikož je počet shluků roven počtu záznamů v datové sadě, respektive každý shluk obsahuje pouze jeden záznam z datové sady, tak centroid každého shluku je roven právě tomu jednomu záznamu v tom shluku a rozdíl mezi nimi je roven nule [34].

Formálně výše popsaná minimalizace lze zapsat jako ( $\vec{\mu}_i$  je centroid  $i$ -tého shluku) [34]:

$$\arg \min \sum_{i=1}^k \sum_{x \in S_i} \|\vec{x} - \vec{\mu}_i\|^2 \quad (6)$$

Celý algoritmus je velmi jednoduchý a má pouze pár kroků [34]:

1. Zvolení umístění  $k$  centroidů v  $n$ -rozměrném prostoru.
2. Ke každé instanci z datové sady najít nejbližší centroid.
3. Přepočítat centroidy na základě přiřazených instancí do shluku.
4. Kroky 2 - 3 opakujeme tak dlouho, dokud se centroid posouvají, respektive dochází k jejich změně v prostoru.

Z výše popsaného algoritmu lze rozpoznat, že se zde nachází několik problémů k vyřešení. Konkrétně se jedná hlavně o tyto následující [34]:

1. Jak velké zvolit  $k$ , respektive kolik chceme shluků.
2. Jak umístit první centroidy do prostoru v inicializační fázi algoritmu.
3. Jakou metriku použít pro zjištění vzdálenosti mezi instancí z datové sady a centroidy.

### 1. Počet $k$ shluků

Způsobů jak zvolit vhodné  $k$  je opět několik. Jedna hlavní a univerzální metoda je spustit K-means vícekrát a vždy s různým  $k$ . Pro každé  $k$  lze vypočítat SSE hodnotu a následně si vykreslit do grafu křivku, která popisuje velikost SSE na základě zvoleného  $k$ . Z grafu následně můžeme vidět bod zlomu, respektive takový bod, od kterého už pro větší  $k$  nedochází k výraznému poklesu SSE hodnoty [34].

### 2. Inicializace centroidů

Zde je opět mnoho způsobů jak zvolit prvotní umístění centroidů v  $n$ -rozměrném prostoru. Nejjednodušší metoda je náhodně vybrat umístění v prostoru. Další je náhodně vybrat instance z datové sady a to považovat za centroidy. Lze taky zvolit podle úplně jiných kritérií, podle znalosti dat [34].

### 3. Metrika

Metrika udává vzdálenost mezi dvěma body v metrickém prostoru. Formálně je metrický prostor je uspořádána dvojice  $(M, d)$ , kde  $M$  je neprázdná množina a  $d$  je metrika, a to je zobrazení  $d : M \times M \rightarrow \mathbb{R}$ , pro které platí, že  $\forall x, \forall y, \forall z \in M$  [20]:

1. Nezápornost, neboli  $d(x, y) \geq 0$ .
2. Totožnost, neboli  $d(x, y) = 0 \Leftrightarrow x = y$ .
3. Symetrie, neboli  $d(x, y) = d(y, x)$ .
4. Trojúhelníková nerovnost, neboli  $d(x, z) \leq d(x, y) + d(y, z)$ .

Existuje mnoho definovaných metrik v různých metrických prostorech. Nechť  $\vec{x}$  a  $\vec{y}$ , jsou vektory z vektorového prostoru  $\mathbb{R}^n$ , potom nejzákladnější metriky jsou:

1. Eukleidovská metrika [20]

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (7)$$

2. Manhattanská metrika [20]

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^n |x_i - y_i| \quad (8)$$

3. Čebyševova metrika [20]

$$d(\vec{x}, \vec{y}) = \max(|x_i - y_i|) \quad (9)$$

### 3.1.6 Evoluční algoritmy

Evoluční algoritmy slouží hlavně k řešení optimalizačních problémů. Používají se tam, kde jsou klasické metody optimalizace nemožné, nebo časově náročné. Algoritmy jsou založeny na základních principech Darwinovy teorie evoluce. Hlavní myšlenka teorie evoluce je v předávání rodičovského genu svým potomkům a tím vytvoření nové populace. Mnoho algoritmů je inspirováno procesy, které můžeme nalézt v přírodě, například: chování mravenců v koloniích nebo chování ptactva při letu. Evoluční algoritmy patří mezi heuristické algoritmy. Evoluční cyklus lze obecně popsat, pro libovolný algoritmus, v následujících bodech [26]:

1. Inicializace parametrů algoritmů, které řídí běh algoritmu. Jedním z parametrů je počet generací, po kterých se algoritmus zastaví. Dále se zde nachází definování účelové funkce. Účelová funkce je matematický popis nebo zápis problému, který je potřeba vyřešit. Následně probíhá minimalizace, nebo maximalizace účelové funkce. Po dosazení parametrů do účelové funkce získáváme hodnotu, která nám říká, jak je dané řešení kvalitní (v závislosti na vstupních parametrech). Často se používá název fitness (vhodnost, kvalita) hodnota, tím se rozumí upravená hodnota účelové funkce, do intervalu  $< 0; 1 >$  [26].
2. Vygenerování první populace, nejčastěji náhodně. Populace obsahuje jedince a jejich počet je dán vstupním parametrem algoritmu. Velikost jedince závisí na řešeném problému, respektive počet parametrů účelové funkce, které se snažíme optimalizovat. Obecně je jedinec reprezentován jako vektor hodnot. Jeden vektor hodnot, respektive jedince, představuje jedno možné řešení účelové funkce. Jedince je předán účelové funkci, která na základě jeho hodnot vrátí hodnotu, která říká, jak je jedinec kvalitní [26].
3. Následně proběhne vyhodnocení každého jedince, z populace, podle účelové funkce.
4. Výběr vhodných rodičů, kteří budou generovat nové potomky.
5. Křížení vybraných rodičů a tím vygenerování nových potomků.
6. Mutace nově vytvořených potomků.
7. Ohodnocení kvality nových potomků, přes účelovou funkci.
8. Zvolení nejvhodnějších jedinců do nové populace, jak z množiny nových potomků, tak z množiny rodičů.

9. Opakuje se vše znova od kroku 4.

Podle způsobu fungování, chování a vlastností algoritmů, je lze rozdělit na [26]:

1. Enumerativní - dochází k výpočtu všech kombinací parametrů účelové funkce. Velmi časově náročné při složitější účelové funkci.
2. Deterministické - používají se metody matematiky, které poskytují požadované výsledky, ale mají omezující předpoklady. Například: účelová funkce je souvislá, problém je lineární a konvexní a další.
3. Stochastické - generují se pouze náhodné hodnoty parametrů účelové funkce. Jsou pomalé a nepřesné.
4. Smíšené - kombinace deterministických a stochastických metod. Poskytují velmi dobré výsledky, jsou robustní a rychlé.

### 3.1.6.1 Genetický algoritmus

Genetický algoritmus je jeden z velmi známých. Algoritmus poskytuje velmi dobré výsledky a patří spíše k těm lepším. Jak už název napovídá, je založen na základních technikách napodobující evoluci, a to: mutaci, křížení a výběr [26].

Genetický algoritmus je zde použit pro řešení problému známého pod názvem Problém obchodního cestujícího (Travelling salesman problem). Tento problém je velmi známý, takže ve zkratce se jedná o nalezení nejkratší cesty mezi zadanými body. Například nejkratší cesta mezi hlavními městy všech států USA. Problém patří do NP-těžkých problémů.

Při řešení problému obchodního cestujícího představuje jednice v populaci vektor čísel, kde každé číslo reprezentuje identifikátor města, respektive bodu. Velikost vektoru je právě rovna počtu bodů  $n$ . Jedinec tedy představuje určitou permutaci průchodu přes všechny body. Celá populace potom obsahuje  $m$  různých jedinců (permutací bodů). Hlavní kroky genetického algoritmu jsou vždy stejné, pouze se mění způsoby, jak se co provádí, podle řešeného problému.

#### 1. Inicializace

Vytvoří se nultá populace, která obsahuje  $m$  jedinců, kteří obsahují náhodné permutace  $n$  bodů.

#### 2. Výběr rodičů

Z populace se vyberou vhodní jedinci, kteří budou tvořit rodiče pro reprodukci nových potomků. Možností jak vybrat rodiče je několik například: ruletový výběr, elitismus, náhodně nebo jiný vlastní způsob, podle řešeného problému [26].

#### 3. Křížení

Křížení znamená prohození částí chromozomů rodičů. Jsou tři hlavní způsoby křížení: jednobodové, dvoubodové a vícebodové. Při jednobodovém křížení dochází k řezu chromozomů obou rodičů ve stejném místě. Nový potomek obsahuje první část chromozomu z prvního rodiče a druhou část chromozomu z druhého rodiče. Při dvoubodovém křížení dochází ke dvěma řezům. Zde potomek obsahuje první část chromozomu z prvního rodiče, druhou část (respektive prostřední) obsahuje z druhého rodiče a poslední třetí část obsahuje opět z prvního rodiče. Poslední vícebodové křížení je náhodné vybírání jednotlivých bitů z obou rodičů [26].

### 3. Mutace

Při mutaci se pouze zmutuje  $n$ -hodnot v jedinci, například jestli jedinec je binární (vektor obsahuje pouze hodnoty 0 a 1), tak potom dojde k mutaci z 0 na 1 a z 1 na 0 u  $n$ -hodnot. V případě problému obchodního cestujícího se prohodí dvě hodnoty mezi sebou na dvou pozicích. Tím se vytvoří nová permutace [26].

### 4. Výběr populace

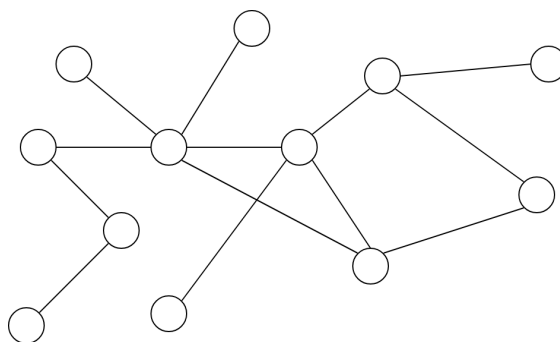
Poslední krok je výběr nových jedinců do nové populace z množiny rodičů a nových potomků vzniklých reprodukcí. Zde účelová funkce představuje vzdálenost celé cesty mezi všemi body. Do nové populace je vybrán buď rodič a nebo jeho potomek na základě vyhodnocení účelové funkce obou jedinců. Lepší postupuje do nové populace, horší umírá.

### 5. Opakování

Vše se znova opakuje od kroku 2, dokud se neprovede předem definovaný počet generací.

#### 3.1.7 Teorie grafů

Teorie grafů se zabývá zkoumáním grafu, což je matematická struktura, která se skládá ze dvou množin. První množina je množina vrcholů a druhá je množina hran. Každá hrana propojuje právě dva vrcholy. Hrana může taky obsahovat váhu. To může například znamenat počet kilometrů mezi dvěma městy (vrcholy). Grafy se reprezentují i graficky a to tak, že vrchol představuje kolečko a hrana úsečku mezi dvěma vrcholy (Obrázek 6) [38].



Obrázek 6: Ukázka konkrétního neorientovaného grafu

### 3.1.7.1 Definice

Formálně je graf  $G$  definovaný jako uspořádaná dvojice, kde  $V$  je množina vrcholů a  $E$  je množina hran [38]:

$$G = (V, E) \quad (10)$$

Graf může být dvojího typu:

1. Orientovaný graf - je takový graf, kde hrany mají přesně definovaný počáteční vrchol a koncový vrchol. Potom množina hran obsahuje uspořádané dvojice binární relace nad množinou  $V$ , neboli  $E \subseteq V \times V$  [38].
2. Neorientovaný graf - je takový graf, kde hrany nemají definovaný počáteční vrchol a koncový vrchol. Potom množina hran obsahuje dvouprvkové množiny obsahující vrcholy. Neboli  $E \subseteq \{\{u, v\} | u, v \in V, u \neq v\}$  [38].

Výpočetní jádro přímo neobsahuje možnost uživateli pracovat s grafem, nicméně vykreslení konečného automatu je založeno na grafech. Mimo to aplikace obsahuje animaci založenou na teorií grafů. Animace bude rozebrána dále v textu.

### 3.1.7.2 Základní vlastnosti a typy grafů

1. Stupeň - počet hran, které patří danému vrcholu. Nechť  $u$  je vrchol, potom  $\deg(u)$  je stupeň vrcholu [38].
  - (a) Neorientovaný graf -  $\deg(u) = |\{e \in E | u \in e\}|$
  - (b) Orientovaný graf
    - i. Vstupní vrchol -  $\deg^+(u) = |\{e \in E | \exists v \in V : e = (v, u)\}|$
    - ii. Výstupní vrchol:  $\deg^-(u) = |\{e \in E | \exists v \in V : e = (u, v)\}|$
2. Multigraf - může obsahovat více hran mezi stejnými vrcholy [38].
3. Sled - posloupnost vrcholů, ve které je mezi každými po sobě jdoucími vrcholy hrana [38].
4. Tah - sled, ve kterém se neopakují hrany [38].
5. Sled - tah, ve kterém se neopakují vrcholy [38].
6. Souvislost - graf je souvislý, právě tehdy když, mezi každými dvěma vrcholy existuje cesta [38].
7. Kružnice - uzavřená posloupnost vrcholů [38].

8. Strom - neorientovaný a souvislý graf, který neobsahuje kružnici. Každé dva vrcholy jsou propojeny právě jednou cestou. Po odebrání jedné libovolné hrany se ze stromu stává nesouvislý graf. Po přidání jedné hrany mezi libovolné hrany vznikne kružnice [38].
9. Úplný graf - neorientovaný graf, kde každé dva rozdílné vrcholy jsou propojeny hranou. Takový graf o  $n$  vrcholech má právě  $\frac{n \cdot (n-1)}{2}$  hran [38].

### 3.1.8 Game of Life

Game of Life, neboli Hra života, je založena na buněčném automatu, určena pro nula hráčů. Z toho vyplývá, že hra je určena počátečním stavem hry a dále už není možný žádný uživatelský vstup. Hra je určena pouze pro sledování vývoje hry. Hra se odehrává ve dvojrozměrné ortogonální mřížce čtvercových buněk, kde každá buňka je právě v jednom ze dvou stavů. Je buď živá nebo mrtvá [53].

#### 3.1.8.1 Pravidla

Celá hrací plocha, respektive její buňky, se dostanou z jednoho stavu do druhého podle pravidel. Změna stavu probíhá na celé hrací ploše pro všechny buňky najednou, změna je v čase diskrétní. Změna stavu celé hrací plochy může být označena jako uplynutí jedné generace. Pravidla jsou definována podle toho, jaké buňky má daná buňka kolem sebe (sousedy) v mřížce [53].

1. Jakákoliv živá buňka s méně než dvěma živými sousedy zemře, nepřežije do další generace.
2. Jakákoliv živá buňka se dvěma nebo třemi živými sousedy zůstává žít, přežije do další generace.
3. Jakákoliv živá buňka s více než třemi živými sousedy zemře, nepřežije do další generace.
4. Jakákoliv mrtvá buňka s právě třemi živými sousedy oživne, oživne v další generaci.

Hra nemá žádné výukové zaměření ve výsledném systému. Je zde pouze ze zajímavého vizuálního hlediska a pro možné pobavení.



## 3.2 Funkce, operace

Tato část se zabývá popisem algoritmů, řešení, postupů a jiných podpůrných vlastností. Pro každou oblast, respektive partii, výpočetní jádro poskytuje základní operace nebo funkce, které uživatel má možnost využít. Například v konečných automatech převod z nedeterministického končného automatu na deterministický konečný automat. Zde budou popsány pouze ty nejzajímavější akce, které stojí za zmínku.

### 3.2.1 Matice

Matice kromě základních operací jako je sčítání, odečítání a násobení obsahují možnost provést Gaussova eliminační metodu a Gauss-Jordánovu eliminační metodu.

#### 3.2.1.1 Gaussova eliminační metoda

Tato metoda provádí řádkové operace tak, aby po dokončení algoritmu byly pod hlavní diagonálou nuly (Algoritmus 1). Matice se převádí do schodového tvaru. Na konci jsou pouze nenulové (mohou být i nulové) čísla v horní trojúhelníkové oblasti matice [52].

---

**Algoritmus 1** Gaussova eliminační metoda

---

```
1:  $A \leftarrow \text{matrix}(m \times n)$ 
2:  $h \leftarrow 1$  ▷ pivot of row
3:  $k \leftarrow 1$  ▷ pivot of column
4: while  $h \leq m$  and  $k \leq n$  do
5:    $i_{max} \leftarrow \arg \max(i = \{h \dots m\}, \text{abs}(A[i, k]))$ 
6:   if  $A[i_{max}, k] = 0$  then
7:      $k \leftarrow k + 1$  ▷ go to the next column
8:   else
9:      $\text{swap rows}(h, i_{max})$ 
10:    for all  $i$  in  $\{h + 1 \dots m\}$  do ▷ for all rows below pivot
11:       $f \leftarrow \frac{A[i, k]}{A[h, k]}$ 
12:       $A[i, k] \leftarrow 0$ 
13:      for all  $j$  in  $\{k + 1 \dots n\}$  do ▷ for all remaining elements in row
14:         $A[i, j] \leftarrow A[i, j] - A[h, j] \cdot f$ 
15:      end for
16:    end for
17:     $h \leftarrow h + 1$ 
18:     $k \leftarrow k + 1$ 
19:  end if
20: end while
```

---

### 3.2.1.2 Gauss-Jordánová eliminační metoda

Tato metoda rozšiřuje Gaussovu eliminační metodu. Dojde k vynulování čísel i v horní trojúhelníkové oblasti. Po vynulování se ještě na konci podělí všechny řádky jejich pivoty, aby byly pivoty rovny 1, respektive čísla na hlavní diagonále. Jinými slovy se matice převede přes řádkové úpravy na jednotkovou matici (identitu). Jestli daná matice řeší soustavu rovnic, tak po aplikaci Gauss-Jordánové eliminační metodě, jsou hned známé hodnoty neznámých. Algoritmus vychází z předchozího algoritmu pro Gaussovu eliminační metodu. Pouze na konci jsou přidány nové operace. Tyto nové operace zde budou pouze popsány (Algoritmus 2) [52].

---

**Algoritmus 2** Gauss-Jordánová eliminační metoda

---

```
1: ... Gaussian elimination
2: for all  $i$  in  $\{h...m\}$  do
3:   for all  $j$  in  $\{k...n\}$  do
4:      $A[i, j] \leftarrow \frac{A[i, j]}{A[h, k]}$ 
5:   end for
6: end for
7: for all  $i$  in  $\{1...m\}$  do
8:   for all  $j$  in  $\{0...i\}$  do
9:      $f \leftarrow A[j, i]$ 
10:    for all  $p$  in  $\{i...n\}$  do
11:       $A[j, p] \leftarrow A[j, p] - A[i, p] \cdot f$ 
12:    end for
13:   end for
14: end for
```

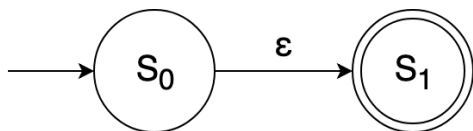
---

### 3.2.2 Regulární jazyk - převod na nedeterministický konečný automat

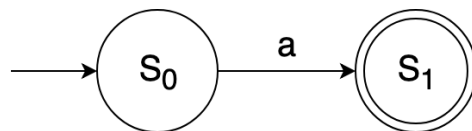
Jednou ze základních operací, která jde udělat s regulárním jazykem, je převod na nedeterministický konečný automat. Převod je velmi jednoduchý a probíhá na základě operací (zřetězení, iterace a sjednocení). Každá operace má definované, jak vypadá k ní požadovaný konečný automat. Jakmile je regulární výraz v syntaktickém stromu, tak následně je možné jednoduchým průchodem do hloubky a typem postorder, vyhodnotit výraz tak, že se postupně z listů stromu skládají malé konečné automaty, na které se postupně při průchodu stromem skládají další konečné automaty. Vždy je dodržena podmínka, že je jeden počáteční stav a jeden přijímací. Na konci celého průchodu je jeden velký nedeterministický konečný automat [27].

#### 3.2.2.1 Primitivní základní automaty

Existují dva základní automaty. Automat pro prázdné slovo (Obrázek 7) a pro libovolný jeden symbol z abecedy automatu (Obrázek 8) [27].



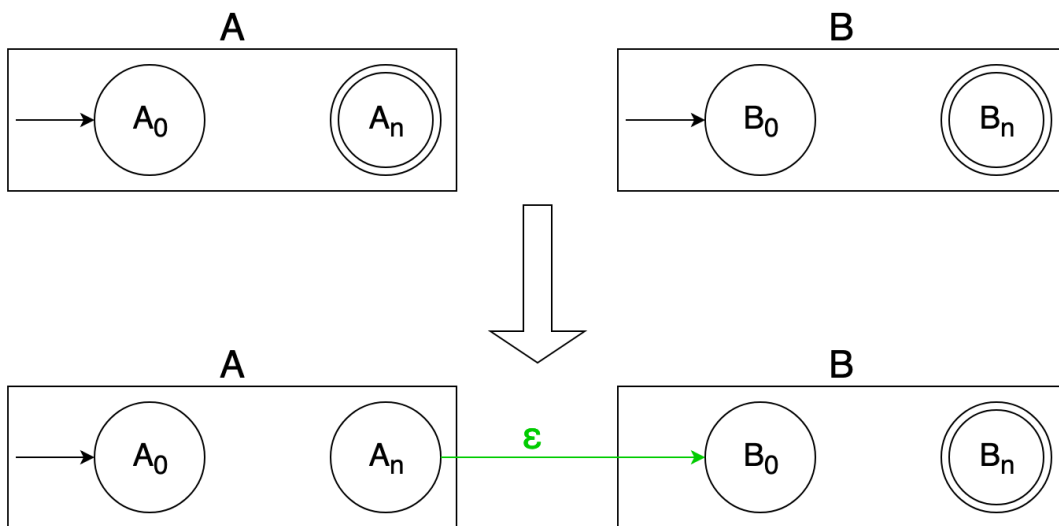
Obrázek 7: Jednoduchý konečný automat pro prázdné slovo  $\epsilon$



Obrázek 8: Jednoduchý konečný automat pro symbol  $a$

### 3.2.2.2 Automat pro zřetězení

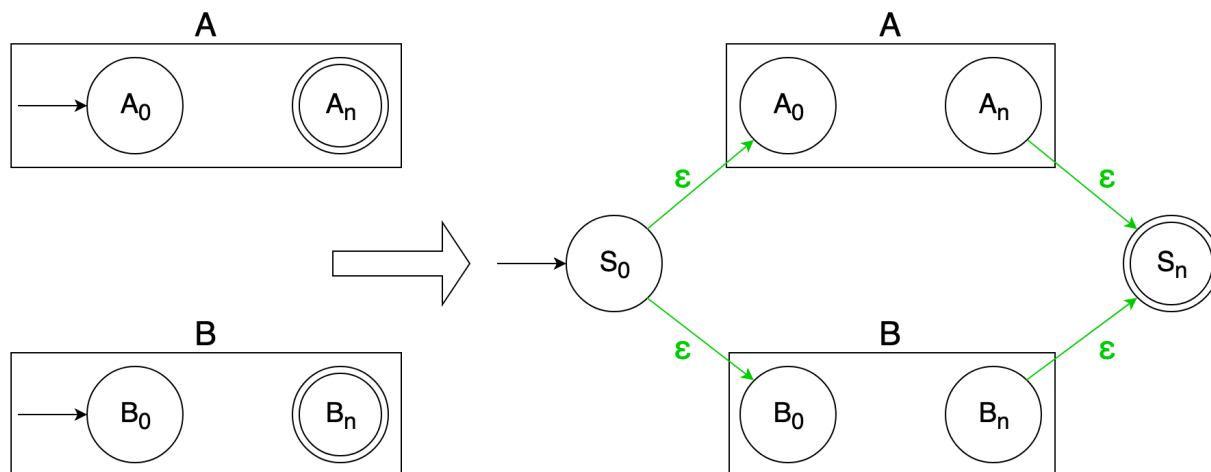
Jakmile jsou primitivní automaty vytvořené, je možné začít na ně aplikovat operace. Nejednodušší operace je zřetězení (Obrázek 9). Při zřetězení dvou automatů, stačí umístit  $\epsilon$  přechod z přijímacího stavu prvního automatu do počátečního stavu druhého automatu [27].



Obrázek 9: Konečný automat pro zřetězení dvou automatů

### 3.2.2.3 Automat pro sjednocení

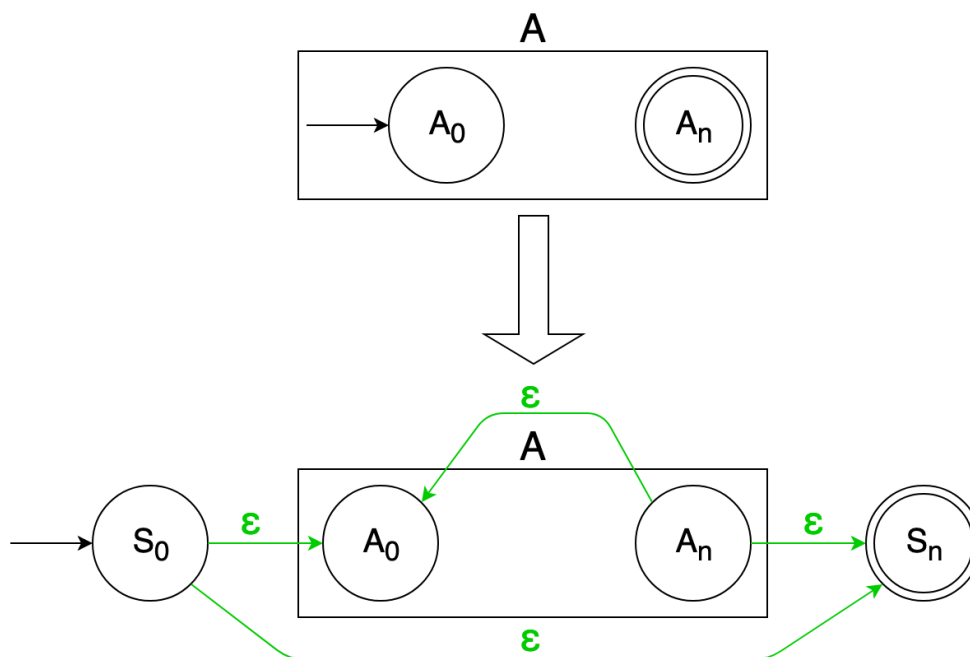
Další operace je sjednocení (Obrázek 10). Zde se vytvoří nový počáteční stav a zároveň nový přijímací stav. Z nového počátečního stavu povedou  $\epsilon$  přechody do počátečních stavů původních automatů. Z přijímacím stavem to je velmi podobně. Do nového přijímacího stavu povedou  $\epsilon$  přechody z přijímacích stavů z původních automatů [27].



Obrázek 10: Konečný automat pro sjednocení dvou automatů

#### 3.2.2.4 Automat pro iteraci

Poslední operace, která zbývá je iterace (Obrázek 11). Tato operace probíhá pouze na jednom automatu. Částečně je operace podobná sjednocení, jelikož zde taky dochází k přidání nového počátečního a přijímacího stavu. Analogicky jsou přidány i  $\epsilon$  přechody z a do nově vytvořených stavů. Zde jsou navíc ještě dva nové  $\epsilon$  přechody. První přechod je z nového přijímacího stavu do přímo přijímacího stavu. Druhý přechod je z přijímacího stavu z původního automatu do počátečního stavu z původního automatu [27].



Obrázek 11: Konečný automat pro iteraci automatu

### 3.2.3 Konečný automat

Pro konečné automaty jsou zde dostupné dvě akce. První je převod nedeterministického konečného automatu na deterministický konečný automat. Druhá akce je minimalizace deterministického konečného automatu. A samozřejmě taky simulace.

#### 3.2.3.1 Převod na deterministický konečný automat

Libovolný nedeterministický konečný automat lze převést na deterministický konečný automat. Převod můžeme provést pomocí tabulky, kde v řádcích budou nové stavy převedeného automatu a ve sloupcích budou symboly z abecedy automatu [27].

Na začátku se vytvoří nová množina, která bude obsahovat všechny počáteční stavy automatu. Tato množina bude umístěna do tabulky na první řádek a následuje vyplnění hodnot pro tento řádek pro každý sloupec (symbol) v tabulce. Pro každý symbol, respektive sloupec, a pro každý stav, který je obsažen v množině na daném řádku, se provede následující operace. Zjistí se, do kterých stavů lze přejít ze všech stavů, které obsahuje množina na daném řádku, a na základě symbolu v daném sloupci. Výsledné zjištění všech stavů se vloží do množiny a ta je umístěna na příslušný řádek a sloupec v tabulce. Jestli tabulka neobsahuje ještě řádek, který má stejnou množinu stavů, tak je množina zároveň umístěna do nového řádku. Po zpracování všech symbolů pro daný řádek, se přejde na další dostupný řádek a probíhá opět znova to stejné, tak dlouho, dokud nezbyvá žádný další řádek. Na konci algoritmu stačí místo množin umístit čísla stavů, v takovém pořadí, v jakém byly přidány [27].

Pro následující pseudokód (Algoritmus 3) je potřeba definovat  $\epsilon - closure(S)$  funkci. Funkce (neboli uzávěr) vrací množinu stavů, na základě vstupní množiny stavů  $S$ , respektive množinu stavů, do kterých se dostane ze stavů z množiny  $S$  přes  $\epsilon$  symbol (přechodovou funkci). Dále bude použita klasická přechodová funkce  $\delta(q, a)$ , ale místo  $q$  se použije množina  $S$ , takže  $\delta(S, a)$ . Funkce vrátí množinu stavů, ze kterých se dostane ze stavů z množiny  $S$  přes symbol  $a$  [25].

---

**Algoritmus 3** Převod NFA na DFA

---

```
1:  $DFA \leftarrow \text{init } DFA$ 
2:  $DFA_{q_0} \leftarrow \epsilon - \text{closure}(\{q_0\})$ 
3:  $S_{ALL} \leftarrow \{\epsilon - \text{closure}(\{q_0\})\}$ 
4:  $F_{ALL} \leftarrow \{\}$ 
5: while  $|S_{ALL}| \neq |F_{ALL}|$  do
6:    $A \leftarrow \text{any set from } S_{ALL}$ 
7:    $F_{ALL} \leftarrow F_{ALL} \cup A$ 
8:   for all  $a$  in  $\Sigma$  do
9:      $S \leftarrow \epsilon - \text{closure}(\delta(A, a))$ 
10:    if  $S \notin S_{ALL}$  then
11:       $S_{ALL} \leftarrow S_{ALL} \cup S$ 
12:    end if
13:     $DFA_\delta \leftarrow DFA_\delta \cup \{((\delta(A, a), a), S)\}$ 
14:  end for
15: end while
16:  $DFA_Q \leftarrow S_{ALL}$ 
17: for all  $S$  in  $S_{ALL}$  do
18:   for all  $s$  in  $S$  do
19:     if  $s \in F$  then
20:        $DFA_F \leftarrow DFA_F \cup \{S\}$ 
21:     end if
22:   end for
23: end for
```

---

### 3.2.3.2 Minimalizace

Každý automat může obsahovat stavy, které jsou nadbytečné, nedosažitelné nebo ekvivalentní, takový automat není minimalizovaný. Každý deterministický konečný automat lze minimalizovat. Jelikož každý nedeterministický konečný automat lze převést na deterministický konečný automat, tak lze po převodu minimalizovat i nedeterministický automat [27].

Minimalizace probíhá ve dvou základních krocích:

1. Eliminace nedosažitelných stavů.
2. Sjednocení ekvivalentních stavů.

#### 1. Eliminace nedosažitelných stavů

Eliminace nedosažitelných stavů je velmi jednoduchá operace. Jedná se o odstranění stavů, takových do kterých se nejde dostat z počátečního stavu. Jinými slovy, neexistuje žádná orientovaná cesta v grafu, která by vedla z počátečního stavu do požadovaného [27].

Algoritmus začíná v počátečním stavu automatu  $q_0$ . Stav  $q_0$  je uložen do množiny dosažitelných stavů. Potom následuje přechod do všech dostupných stavů, do kterých je možné se dostat ze stavu  $q_0$ . Navštívené stavy se vloží na zásobník. Jakmile se projdou všechny přechody stavu  $q_0$ , tak se stav uloží množiny zpracovaných stavů. Následně se celá akce opakuje z dalším stavem, který je v zásobníku, respektive čeká na zpracování. Algoritmus běží tak dlouho, dokud zásobník není prázdný (Algoritmus 4) [27].

---

**Algoritmus 4** Eliminace nedosažitelných stavů

---

```

1:  $R \leftarrow \{q_0\}$ 
2:  $W \leftarrow \{q_0\}$ 
3: while  $W$  is not empty do
4:    $N \leftarrow \{\}$  ▷ Empty set
5:   for all  $q$  in  $W$  do
6:     for all  $a$  in  $\Sigma$  do
7:       if  $\delta(q, a)$  then
8:          $N \leftarrow N \cup \{\delta(q, a)\}$ 
9:       end if
10:    end for
11:  end for
12:   $W \leftarrow N \setminus R$  ▷ set difference
13:   $R \leftarrow R \cup W$ 
14: end while

```

---

## 2. Sjednocení ekvivalentních stavů

Hledání ekvivalentních stavů probíhá na základě rozdělení stavů do tříd ekvivalence. Předpoklad je, že automat už nemá žádné nedosažitelné stavy. Výpočet si lze představit za použití tabulky, kde v řádcích jsou stavy automatu a ve sloupcích všechny symboly z abecedy automatu.

Na začátku jsou všechny stavy rozděleny do dvou tříd. První třída obsahuje stavy, které jsou nepřijímající a druhá třída obsahuje stavy, které jsou přijímající. Následně se do tabulky, do každého řádku a sloupce, vyplní třída, na základě stavu (řádek tabulky) a symbolu (sloupec tabulky). Jinými slovy se vyplní na požadované místo v tabulce taková třída, do které se jde dostat z daného stavu a daného symbolu. Respektive se dostane do stavu, ale každý stav patří do nějaké třídy ekvivalence, do kterých byly stavy už na začátku rozděleny. Po vyplnění všech přechodů a k nim příslušné třídy ekvivalence se zjistí, jestli jsou všechny stavy v dané třídě ekvivalence stejné. To se rozpozná tak, že pro každý stav v dané třídě a pro každý symbol (respektive přechod ze stavu na základě symbolu) musí patřit výsledný stav do stejné třídy ekvivalence. Jestli se některé stavy (respektive třídy ekvivalence) v nějakém symbolu neshodují, tak se daná třída musí rozpadnout na dvě třídy. A tím vznikne nová třída ekvivalence. Potom

se opět vyplní celá tabulka a vše pokračuje znova, tak dlouho, dokud už nelze žádnou třídu ekvivalence rozdělit. Na konci algoritmu jsou všechny stavy umístěny do  $n$ -tříd ekvivalence. Jestliže libovolná třída obsahuje více než jeden stav, je možné tyto stavy sloučit dohromady. Tím získáme redukovaný konečný automat. Pro algoritmizaci sjednocení ekvivalentních stavů byl použit známý Hopcroft algoritmus (Algoritmus 5) [4].

---

**Algoritmus 5** Sjednocení ekvivalentních stavů (Hopcroft)

---

```

1:  $P \leftarrow \{F, Q \setminus F\}$  ▷ set difference
2:  $W \leftarrow \{F\}$ 
3: while  $W$  is not empty do
4:    $A \leftarrow$  any element from  $W$ 
5:    $W \leftarrow W \setminus A$ 
6:   for all  $a$  in  $\Sigma$  do
7:      $X \leftarrow \{\}$ 
8:     for all  $s$  in  $Q$  do
9:       if  $\delta(s, a) \in A$  then
10:         $X \leftarrow X \cup \{s\}$ 
11:      end if
12:      for all  $Y$  in  $P$  do
13:        if  $X \cap Y = \emptyset$  or  $Y \setminus X = \emptyset$  then
14:          continue
15:        end if
16:         $P \leftarrow P \setminus Y$ 
17:         $P \leftarrow P \cup (X \cap Y) \cup (Y \setminus X)$ 
18:        if  $Y \in W$  then
19:           $W \leftarrow W \setminus Y$ 
20:           $W \leftarrow W \cup (X \cap Y) \cup (Y \setminus X)$ 
21:        else
22:          if  $|X \cap Y| \leq |Y \setminus X|$  then
23:             $W \leftarrow W \cup (X \cap Y)$ 
24:          else
25:             $W \leftarrow W \cup (Y \setminus X)$ 
26:          end if
27:        end if
28:      end for
29:    end for
30:  end for
31: end while

```

---



### 3.2.3.3 Simulace

Aby bylo možné ověřit, jestli konečný automat přijímá, nebo nepřijímá uživatelem zadaný vstup (slovo), tak se musí provést simulace automatu (Algoritmus 6). Simulace začíná v počátečním stavu a postupně čte jeden symbol za druhým ze vstupu. Na základě přečtených symbolů dochází ke změně stavu automatu. Jakmile dojde k přečtení celého slova ze vstupu, ověří se, že automat skončil alespoň v jednom z přijímacích stavů. Jestli ano, automat úspěšně přijal slovo, jinak to končí neúspěchem.

---

**Algoritmus 6** Simulace konečného automatu

---

```
1: procedure SIMULATE( $s$ ,  $input$ )
2:    $S \leftarrow \{\}$ 
3:   if  $|input| > 0$  then
4:      $a \leftarrow$  get and remove first symbol from  $input$ 
5:      $T \leftarrow \{\delta(s, a) \mid (s, a) \in \delta\} \cup \{\delta(s, a) \mid (s, \epsilon) \in \delta\}$ 
6:     for all  $t$  in  $T$  do
7:        $S \leftarrow S \cup simulate(t, input)$ 
8:     end for
9:   else
10:     $S \leftarrow S \cup \{s\}$ 
11:  end if
12:  return  $S$ 
13: end procedure
14: if  $\exists s \in simulate(q_0, input) : s \in F$  then
15:    $print(success)$ 
16: end if
```

---

### 3.2.4 Bezkontextové gramatiky - redukce

Redukce gramatiky odstraňuje přebytečné neterminály. Jelikož bezkontextová gramatika může obsahovat neterminály, které nejsou dosažitelné ze startovního neterminálu. Stejně tak může existovat bezkontextová gramatika, která negeneruje žádné slovo. Každá bezkontextová gramatika lze zredukovat [27].

Redukce se skládá ze dvou hlavních kroků, které musí být provedeny ve správném pořadí [27]:

1. Odstranění neterminálů, které negenerují terminální slovo.
2. Odstranění nedosažitelných neterminálů.

### 3.2.4.1 Odstranění neterminálů, které negenerují terminální slovo

Na začátku se projdou pravidla a přidají se neterminály do množiny  $\tau$ , takové které generují terminální slovo. Následně se znova procházejí pravidla a zjišťuje se, jestli je možné libovolný další neterminál přepsat na terminál pomocí neterminálů z množiny  $\tau$ . Jestli ano, tak se daný neterminál přidá do množiny  $\tau$ . Celé se to opakuje dokud dochází k přidávání neterminálů do množiny  $\tau$ . Na konci se zkontroluje, jestli množina  $\tau$  obsahuje počáteční neterminál. Jestli počáteční neterminál není obsažen, tak redukce končí a zadaná gramatika není schopna vygenerovat ani jedno slovo (Algoritmus 7) [27].

---

**Algoritmus 7** Odstranění neterminálů, které negenerují terminální slovo

---

```
1:  $\tau \leftarrow \{\}$ 
2:  $l \leftarrow -1$ 
3: while  $|\tau| \neq l$  do
4:   for all  $v$  in  $V$  do
5:     for all  $r$  in  $R(v)$  do
6:       for all  $r_i$  in  $r$  do
7:         if  $\forall a \in r_i : a \in \Sigma$  then
8:            $\tau \leftarrow \tau \cup \{v\}$ 
9:         else if  $\exists a \in r_i : a \in \tau$  then
10:           $\tau \leftarrow \tau \cup \{v\}$ 
11:        end if
12:      end for
13:    end for
14:  end for
15:   $l \leftarrow |\tau|$ 
16: end while
```

---

### 3.2.4.2 Odstranění nedosažitelných neterminálů

Z předchozího bodu je gramatika už částečně redukováná, zbývá poslední krok a to odstranit všechny nedosažitelné neterminály (Algoritmus 8). Do množiny  $W$  se umístí počáteční neterminál. Následně se prochází pravidla daného neterminálu a zjišťuje se, na jaké neterminály lze daný neterminál přepsat. Jakmile se narazí na nový neterminál, který není v množině  $D$ , přidá se do množiny  $W$ . Po projití všech pravidel daného neterminálu, dojde k odstranění neterminálu z množiny  $W$  a vložení do množiny  $D$ . Následně se celá operace opakuje, dokud množina  $W$  není prázdná [27].

---

**Algoritmus 8** Odstranění nedosažitelných neterminálů

---

```
1:  $D \leftarrow \{\}$ 
2:  $W \leftarrow \{S\}$ 
3: while  $W$  is not empty do
4:    $v \leftarrow$  any element from  $W$ 
5:   for all  $r$  in  $R(v)$  do
6:     for all  $r_i$  in  $r$  do
7:       if  $\exists a \in r_i : a \in V \wedge a \notin D$  then
8:          $W \leftarrow W \cup \{a\}$ 
9:       end if
10:    end for
11:  end for
12:   $W \leftarrow W \setminus \{v\}$ 
13:   $D \leftarrow D \cup \{v\}$ 
14: end while
```

---

### 3.2.5 Machine learning - K-means

K-means je velmi jednoduchý (Algoritmus 9). První se náhodně zvolí  $k$  centroidů. Následně se spočítají vzdálenosti pro každý element z data setu a přiřadí se k nejbližšímu clusteru. Potom se pro každý cluster přepočítá jeho střed, podle elementů přiřazených do clusteru z předchozího kroku. Algoritmus končí, jakmile se centroidy clusterů dále nemění [34].

---

**Algoritmus 9** K-means

---

```
1:  $change \leftarrow true$ 
2: while  $change$  do
3:   for all  $element$  in  $dataset$  do
4:      $cluster_i \leftarrow cluster_i \cup \arg \min(distance(cluster_i, element))$ 
5:   end for
6:    $change \leftarrow false$ 
7:   for all  $cluster$  in  $clusters$  do
8:      $average \leftarrow \text{init Cluster}$ 
9:      $average_i \leftarrow \sum_{e \in cluster.elements} e.position_i$ 
10:     $average_i \leftarrow \frac{average_i}{|cluster.elements|}$ 
11:    if  $average \neq cluster.centroid$  then
12:       $change \leftarrow true$ 
13:    end if
14:  end for
15: end while
```

---

### 3.2.6 Evoluční algoritmy - Genetický algoritmus

To jak funguje genetický algoritmus a jeho kroky, bylo rozebráno v teoretické části. Tady bude rozepsán pouze pseudokód. Algoritmus je rozdělen na tři části. Hlavní tok (Algoritmus 11), křížení (Algoritmus 12) a mutace (Algoritmus 10).  $d$  znamená dimenze problému. U TSP je  $d$  rovno počtu měst.

---

**Algoritmus 10** Genetický algoritmus - Mutace

---

```
1: procedure mutate(individual)
2:    $i \leftarrow \text{random}(\{0\dots d\})$ 
3:    $j \leftarrow \text{random}(\{0\dots d\} \setminus \{i\})$ 
4:    $\text{swap} \leftarrow \text{individual}(i, j)$ 
5: end procedure
```

---

---

**Algoritmus 11** Genetický algoritmus - Hlavní tok

---

```
1:  $ml \leftarrow \text{mutation limit}$ 
2:  $pop \leftarrow \text{random population}$ 
3:  $gs \leftarrow \text{generations count}$ 
4:  $b \leftarrow \text{best individual}$ 
5: while  $g$  in  $0\dots gs$  do
6:    $n \leftarrow \text{Array}(|pop|)$ 
7:   for all individual in pop do
8:      $parent \leftarrow \text{random}(pop)$ 
9:      $offspring \leftarrow \text{crossover}(individual, parent)$ 
10:     $m \leftarrow \text{random}(0\dots 1)$ 
11:    if  $m < ml$  then
12:       $offspring \leftarrow \text{mutate}(offspring)$ 
13:    end if
14:    if  $\text{costfunction}(offspring) < \text{costfunction}(individual)$  then
15:       $n \leftarrow \text{append}(offspring)$ 
16:    else
17:       $n \leftarrow \text{append}(individual)$ 
18:    end if
19:    if  $\text{costfunction}(offspring) < \text{costfunction}(b)$  then
20:       $b \leftarrow offspring$ 
21:    end if
22:  end for
23:   $pop \leftarrow n$ 
24: end while
```

---

---

**Algoritmus 12** Genetický algoritmus - Křížení

---

```
1: procedure crossover(parent1, parent2)
2:    $l \leftarrow 0$ 
3:    $u \leftarrow 0$ 
4:    $E \leftarrow \{0, d, d - 1, d - 2\}$ 
5:    $f \leftarrow \text{random}(\{0 \dots d\} \setminus E)$ 
6:    $s \leftarrow \text{random}(\{0 \dots d\} \setminus (E \cup \{f\}))$ 
7:   if  $f < s$  then
8:      $l \leftarrow f$ 
9:      $u \leftarrow s$ 
10:  else
11:     $l \leftarrow s$ 
12:     $u \leftarrow f$ 
13:  end if
14:  offspring  $\leftarrow \text{Array}(d)$ 
15:  offspring[0 :  $l$ ]  $\leftarrow \text{parent}_1[0 : l]$ 
16:   $S \leftarrow \{\text{parent}_1[0 : l], \text{parent}_1[u : d]\}$ 
17:  for all  $i$  in  $l \dots u$  do
18:    if  $\text{parent}_2[i] \notin S$  then
19:      offspring[ $i$ ]  $\leftarrow \text{parent}_2[i]$ 
20:       $S \leftarrow S \cup \{\text{parent}_2[i]\}$ 
21:    end if
22:  end for
23:   $\text{delta} \leftarrow d - |S|$ 
24:  for all  $v$  in parent2 do
25:    if  $v \notin S$  then
26:      offspring[ $|S|$ ]  $\leftarrow v$ 
27:       $S \leftarrow S \cup \{v\}$ 
28:       $\text{delta} \leftarrow \text{delta} - 1$ 
29:      if  $\text{delta} = 0$  then
30:        break
31:      end if
32:    end if
33:  end for
34:  offspring[ $u : d$ ]  $\leftarrow \text{parent}_1[u : d]$ 
35: end procedure
```

---

### 3.2.7 Teorie grafů

Na základě teorií grafů jsou zde založeny dva algoritmy. Vykreslení grafu a animace.

### 3.2.7.1 Vykreslení grafu

Vykreslení grafu se používá pro vykreslení konečného automatu. Je zde použit Force-Directed graph drawing algoritmus, který je založen na systému pružin a elektrických sil (Algoritmus 13). Jedná se o jeden ze základních algoritmů na vykreslení grafů, ale i přesto poskytuje dobré výsledky.

Pružinové síly založené na Hookově zákoně jsou použity k přitáhnutí koncových vrcholů k sobě, pro danou hranu grafu. Odpudivé síly mezi elektrickými nabitými částicemi, které jsou založeny na Coulombově zákoně, jsou použity pro oddělení vrcholů [47].

---

#### Algoritmus 13 Force-directed graph drawing

---

```

1:  $area \leftarrow W \cdot L$  ▷ Width and length of frame
2:  $G \leftarrow (V, E)$ 
3:  $k \leftarrow \sqrt{\frac{area}{|V|}}$ 
4: function  $f_a(x) \leftarrow return \frac{x^2}{k}$ 
5: function  $f_r(x) \leftarrow return \frac{k^2}{x}$ 
6: for all  $i$  in  $0 \dots iterations$  do
7:   for all  $v$  in  $V$  do ▷ Vertex has two vectors:  $.pos$  and  $.disp$ 
8:      $v.disp \leftarrow 0$ 
9:     for all  $u$  in  $V$  do
10:      if  $u \neq v$  then
11:         $delta \leftarrow v.pos - u.pos$ 
12:         $v.disp \leftarrow v.disp + (\frac{\delta}{|\delta|}) \cdot f_r(|\delta|)$ 
13:      end if
14:    end for
15:  end for
16:  for all  $e$  in  $E$  do ▷ Edge has ordered pair of vertices:  $.v$  and  $.u$ 
17:     $delta \leftarrow e.v.pos - e.u.pos$ 
18:     $e.v.disp \leftarrow e.v.disp - (\frac{\delta}{|\delta|}) \cdot f_a(|\delta|)$ 
19:     $e.u.disp \leftarrow e.u.disp + (\frac{\delta}{|\delta|}) \cdot f_a(|\delta|)$ 
20:  end for
21:  for all  $v$  in  $V$  do
22:     $v.pos \leftarrow v.pos + (\frac{v.disp}{|v.pos|}) \cdot \min(v.disp, t)$ 
23:     $v.pos.x \leftarrow \min(\frac{W}{2}, \max(-\frac{W}{2}, v.pos.x))$ 
24:     $v.pos.y \leftarrow \min(\frac{L}{2}, \max(-\frac{L}{2}, v.pos.y))$ 
25:  end for
26:   $t \leftarrow cool(t)$ 
27: end for

```

---

### 3.2.7.2 Animace

Na obrazovce, kde se uživatel přihlašuje, je použita animace, která je založena na teorií grafů (Algoritmus 14). Animace funguje na principu, kde jsou na ploše rozmístěny vrcholy, které putují po přímce na druhou stranu plochy. Když vrchol dosáhne druhé strany plochy, tak se náhodně opět vygeneruje startovní a koncová pozice pro vrchol na ploše. Každý vrchol má vlastní rychlost. Jakmile se vrcholy k sobě dostatečně přiblíží, tak dojde k vykreslení hrany mezi nimi. Naopak jakmile se oddálí opět od sebe, tak hrana zmizí. Toto se opakuje pro všechny vrcholy, které se pohybují po obrazovce, neustále dokola 60x za sekundu. Tato použitá metoda vizuálně působí tak, že to vypadá na ploše v průběhu času, že dochází k vytváření různých tvarů, které se různě pohybují.

---

**Algoritmus 14** Grafová animace

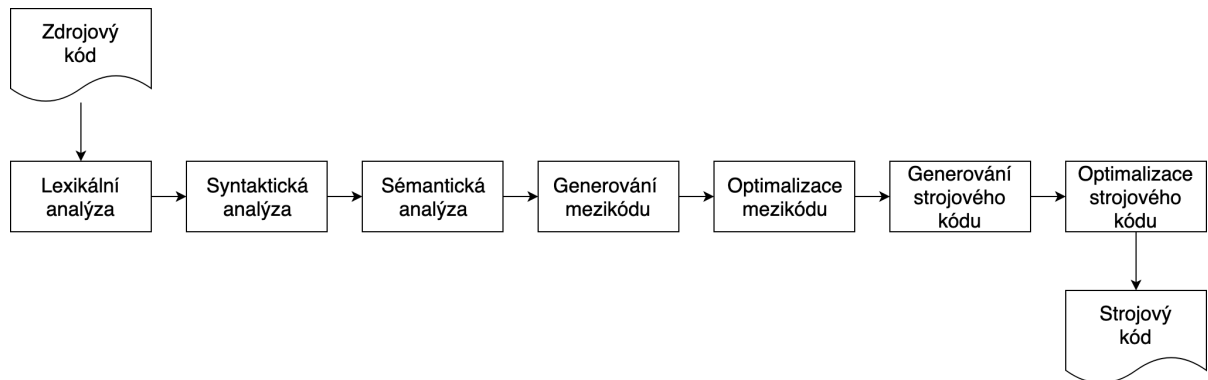
---

```
1:  $dl \leftarrow \text{distance limit}$ 
2: for all  $e$  in  $E$  do
3:    $\text{remove edge}(e)$ 
4: end for
5: for all  $v$  in  $V$  do
6:   for all  $u$  in  $V$  do
7:      $d \leftarrow \text{distance}(u, v)$ 
8:     if  $\text{distance} < dl$  then
9:        $\text{add edge}(v, u)$ 
10:    end if
11:   end for
12: end for
```

---

### 3.3 Syntaktická analýza

Jakmile uživatel zadá libovolný vstup, tak je potřeba daný vstup zkontrolovat, jestli odpovídá požadovanému zápisu. Dále aby bylo možné s uživatelským vstupem provádět další akce, například převod regulárního výrazu na automat, je potřeba vstup z uživatelského rozhraní převést na vhodnou datovou strukturu (Obrázek 13). Tohle všechno a mnohem více zajišťuje syntaktická analýza. Jedná se o jednu z prvních částí překladače (Obrázek 12). Ještě před syntaktickou analýzou je lexikální analýza, která provádí převod vstupu na lexémy. Jelikož zde není potřeba implementovat všechny části překladače, tak je zde popsána pouze syntaktická analýza, která dostatečně pokrývá požadavky vytvářeného systému [33].



Obrázek 12: Kroky překladače

Na začátku je potřeba ověřit, jestli vstup zadaný uživatelem odpovídá předpisu. K zadání předpisu vstupu slouží speciální typ bezkontextové a kontextové gramatiky. Gramatika popisuje celý jazyk, který uživatel může zadat a bude to bráno jako validní vstup. Výsledek syntaktické analýzy je abstraktní syntaktický strom, respektive konkrétní syntaktický strom pro konkrétní vstup, který představuje zadaný výraz ve stromové struktuře. Tato struktura už lze procházet klasickými metodami průchodů stromů, jako je průchod do šířky a hloubky. Taktéž lze strom transformovat a tím například ekvivalentně výraz upravit a zjednodušit, respektive provést optimalizaci výrazu ve stromu.

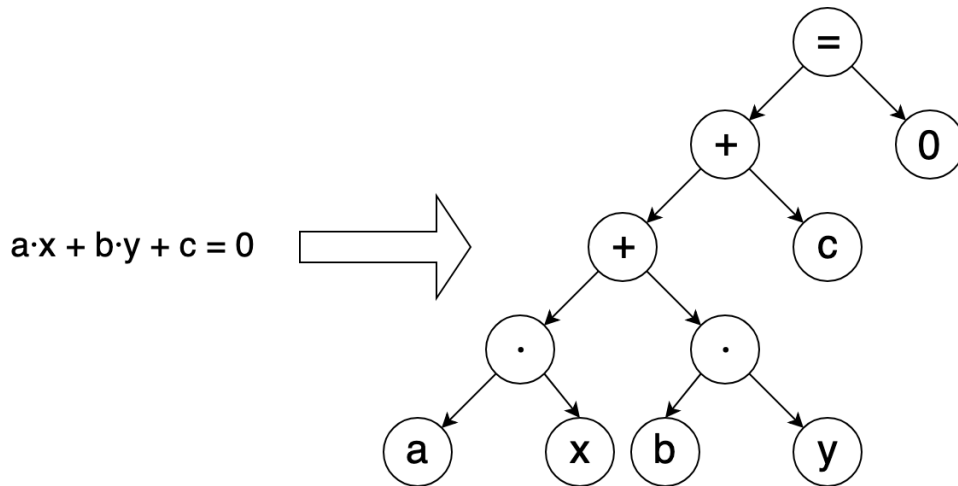
Jsou dva základní způsoby, jak vytvořit abstraktní syntaktický strom [33]:

1. Překlad shora dolů - LL gramatiky (Left to right, Leftmost derivation).
2. Překlad zdola nahoru - LR gramatiky (Left to right, Rightmost derivation).

Jak už názvy napovídají, tak se jedná o způsoby, kterým je výsledný strom vygenerován. První způsob funguje tak, že se začne od startovního neterminálu gramatiky a postupně se snaží derivovat zadaný vstup, respektive dojít k terminálním symbolům. Naopak u překladu zdola



nahoru dochází k tomu, že se začíná u terminálních symbolů a snaží se dojít k startovnímu neterminálu. Pro oba dva zmíněné způsoby existující speciální typy gramatik LL a LR. První písmeno říká, že se čte vstup zleva doprava a druhé písmeno říká, že se používá buď levá derivace nebo pravá derivace.



Obrázek 13: Ukázka převodu výrazu na syntaktický strom

### 3.3.1 Překlad shora dolů

Zde bude rozebrán překlad shora dolů, který je bohatě dostačující na požadavky tohoto systému. Jak už bylo zmíněno výše, tak překlad shora dolů se snaží najít levou derivaci pro zadaný vstup, s tím, že začíná ve startovním neterminálu gramatiky. Pro tento způsob překlada je potřeba, aby gramatika byla typu LL. Tento typ gramatiky bude dále podrobně rozebrán. Následná reálná implementace tohoto způsobu se nazývá jako prediktivní syntaktický analyzátor [33].

### 3.3.2 First a Follow

Ještě předtím než bude vysvětlena LL gramatika, je potřeba si zadefinovat dvě množiny, First a Follow množinu. First množinu vypočítá funkce označovaná jako  $FIRST(\alpha)$  a Follow množinu funkce označovaná jako  $FOLLOW(\alpha)$ , kde  $\alpha$  je vstup tvořený symboly bezkontextové gramatiky  $G$  [33].

#### 3.3.2.1 FIRST

Množina  $FIRST(\alpha)$  obsahuje terminální symboly, které lze získat derivací z řetězce  $\alpha$ . Jinými slovy je to množina terminálů, kterými vstupní řetězec může začínat.

Nechť  $G$  je bezkontextová gramatika a  $\alpha$  je řetězec tvořený symboly gramatiky  $G$ , potom  $FIRST(\alpha)$  je funkce dána následujícím předpisem [33]:

$$FIRST(\alpha) = \{a \in \Sigma | \alpha \Rightarrow^* a\beta, \beta \in (\Sigma \cup V)^*\} \cup \{\epsilon | \alpha \Rightarrow^* \epsilon\} \quad (11)$$

### Algoritmus

Množinu follow lze určit jak pro terminály, neterminály, tak i pro řetězec. Výpočet  $FIRST(\alpha)$  je následující [33]:

1. Jestliže  $\alpha$  je terminální symbol, výsledná množina obsahuje daný terminální symbol a výpočet skončil.
2. Jestliže  $\alpha$  je neterminální symbol, potom:
  - (a) Jestliže  $\alpha$  je pravidlo ve tvaru  $\alpha \rightarrow \epsilon$ , potom se přidá do výsledné množiny prázdný řetězec  $\epsilon$ .
  - (b) Jestliže  $\alpha$  je pravidlo ve tvaru  $\alpha \rightarrow Y_1, Y_2, \dots, Y_k$ , potom se přidá symbol  $a$  do množiny, pokud pro libovolné  $i$  platí, že  $a \in FIRST(Y_i)$  a  $\epsilon$  je ve všech množinách  $FIRST(Y_1), FIRST(Y_2), \dots, FIRST(Y_{i-1})$ .
  - (c) Jestliže  $\epsilon \in FIRST(Y_j)$ , kde  $j = 1, 2, \dots, k$ , potom se přidá do výsledné množiny  $\epsilon$ .
3. Kroky opakujeme pro všechny pravidla, dokud přidáváme nové symboly do množin.

#### 3.3.2.2 FOLLOW

Množina  $FOLLOW(A)$  lze spočítat pouze pro neterminální symboly, takže  $A$  je zde neterminál. Množina  $FOLLOW(A)$  obsahuje terminální symboly, které se mohou bezprostředně objevit vpravo od neterminálu  $A$  [33].

Nechť  $G$  je bezkontextová gramatika a  $A$  je neterminální symbol gramatiky  $G$ , potom  $FOLLOW(A)$  je funkce dána následujícím předpisem [33]:

$$FOLLOW(A) = \{a \in \Sigma \mid S \Rightarrow^* \alpha A \beta, a \in FIRST(\beta), \alpha, \beta \in (\Sigma \cup V)^*\} \quad (12)$$

### Algoritmus

Následující kroky popisují výpočet množiny  $FOLLOW(A)$  pro neterminál  $A$  [33]:

1. Jestliže neterminál  $A$  je startovní neterminál  $S$ , tak se do výsledné množiny se vloží symbol  $\$,$  který značí konec vstupního řetězce.
2. Jestliže  $A$  je pravidlo ve tvaru  $A \rightarrow \alpha B \beta$ , potom se do množiny  $FOLLOW(B)$  umístí obsah z  $FIRST(\beta)$ , kromě  $\epsilon$ .
3. Jestliže  $A$  je pravidlo ve tvaru  $A \rightarrow \alpha B$  nebo  $A \rightarrow \alpha A \beta$ , kde  $FIRST(\beta)$  obsahuje  $\epsilon$ , potom se přidají prvky do množiny  $FOLLOW(B)$  z množiny  $FOLLOW(A)$ .
4. Kroky opakujeme pro všechny pravidla a na všechny neterminální symboly pravých stran pravidel, dokud přidáváme nové symboly do množin.

### 3.3.2.3 Příklad

Pro lepší pochopení je zde uveden ukázkový příklad výpočtů first a follow množin pro danou gramatiku.

Nechť  $G$  je bezkontextová gramatika definována přepisovacíma pravidly:

1.  $E \rightarrow TE'$
2.  $E' \rightarrow +TE' | \epsilon$
3.  $T \rightarrow FT'$
4.  $T' \rightarrow *FT' | \epsilon$
5.  $T \rightarrow (E) | id$

Potom first a follow množiny jsou následující:

- |                                   |                                   |
|-----------------------------------|-----------------------------------|
| • $FIRST(E) = \{ (, id \}$        | • $FOLLOW(E) = \{ ), \$ \}$       |
| • $FIRST(T) = \{ (, id \}$        | • $FOLLOW(E') = \{ ), \$ \}$      |
| • $FIRST(F) = \{ (, id \}$        | • $FOLLOW(T) = \{ +, ), \$ \}$    |
| • $FIRST(E') = \{ +, \epsilon \}$ | • $FOLLOW(T') = \{ +, ), \$ \}$   |
| • $FIRST(T') = \{ *, \epsilon \}$ | • $FOLLOW(F) = \{ +, *, ), \$ \}$ |

### 3.3.3 LL(1) gramatiky

LL(1) gramatika slouží k vytvoření rozkladové tabulky (bude vysvětleno dále). První L znamená, že se vstup čte zleva doprava. Druhé L znamená, že se používá levá derivace a 1 určuje počet symbolů (v tomto případě jeden symbol), které je potřeba znát, aby parser správně rozhodnul, jaké pravidlo gramatiky použít. 1 symbol je dostačující pro většinu jazyků a je použit i v tomto vytvářeném systému.

Nicméně, ne každá gramatika je LL(1). Aby gramatika  $G$  byla typu  $LL(1)$ , musí splňovat následující dvě vlastnosti. Pro každé pravidlo  $A$  a pro každé pravé strany pravidla (řetězce)  $\alpha, \beta$ , kde  $\alpha \neq \beta$ , tak, že  $A \rightarrow \alpha, \beta$ , platí [33]:

1. Množiny  $FIRST$  všech pravých stran musejí být disjunktní. Platí pro všechny dvojice pravých stran mezi sebou.

$$FIRST(\alpha) \cap FIRST(\beta) = \emptyset \quad (13)$$

2. Jestliže  $\alpha \Rightarrow^* \epsilon$ , potom musí být  $FOLLOW(A)$  disjunkt s množinami  $FIRST(\beta)$  zbývajících pravých stran pravidla. Platí pro všechny dvojice pravých stran mezi sebou.

$$\alpha \Rightarrow^* \epsilon : FIRST(\beta) \cap FOLLOW(A) = \emptyset \quad (14)$$

### 3.3.4 Parser

Existují dvě hlavní možnosti, jak implementovat parser, který bude implementován podle LL(1) gramatiky a bude schopen převést vstup na odpovídající výstup, respektive syntaktický strom. Jestliže parser selže, tzn. vstup neodpovídá gramatice, tak dojde k nahlášení chyby.

#### 3.3.4.1 Rekurzivní sestup

První a nejjednodušší možnost implementace je rekurzivní sestup (recursive descent parser). Tato metoda funguje na jednoduchém principu a to tak, že pro každý neterminál se napíše v programu metoda. Jednotlivé těla metod daných neterminálu volají rekurzivně další metody respektive další neterminály [33].

Nechť  $A$  je neterminál a přepisovací pravidlo ve tvaru  $A \rightarrow \alpha, \beta$ , potom tělo metody  $A()$  v programu bude obsahovat analýzu symbolů  $\alpha, \beta$ . Jestli je libovolný symbol z pravé strany přepisovacího pravidla terminál, tak se zjistí, jestli aktuální načtený vstup odpovídá očekávanému terminálu. Pokud symbol neodpovídá, tak vstup neodpovídá gramatice. Jestli je libovolný symbol z pravé strany přepisovacího pravidla neterminál, tak se zavolá jeho metoda [33].

Takhle to probíhá postupně rekurzivně do doby, než je zpracován celý vstup. Jakmile parser zpracuje celý vstup a neskončil s chybou, tak vstup odpovídá gramatice. Při volání jednotlivých metod se postupně buduje syntaktický strom. Metody daných neterminálů se implementují, respektive volají podle vyžadované priority pořadí. Například při parsování aritmetického výrazu s operátory  $+$ ,  $-$ ,  $\cdot$ ,  $a^x$  má sčítání a odčítání nejnižší prioritu, takže se začne od těchto operátorů a postupně se rekurzivním sestupem (volání metod v tělech metodách jednotlivých neterminálů) dojde na operátory s vyšší prioritou. V tomto příkladu to znamená, že se z metod pro sčítání a odčítání zavolá metoda pro násobení a ta zavolá metodu pro mocninu. Po dokončení operací v těle metod se postupně parser vrací v rekurzivním volání zpět nahoru, respektive k první zvolané metodě. Tento postup se opakuje pořád dokola, podle toho jak se čte, respektive parsuje vstup.

Jak lze z výše uvedeného popisu vidět, jedná se o velmi jednoduchý způsob implementace. Tento způsob je vhodný při ruční implementaci.

#### 3.3.4.2 Nerekurzivní prediktivní přístup

Tato metoda se skládá z několika částí: vstup, zásobník, rozkladová tabulka, řídicí program a výstup (Obrázek 14). Vstup je vstupní řetězec ukončený symbolem  $\$$ . Zásobník obsahuje

symbols gramatiky, kde jako první hodnota v zásobníku, tedy úplně dole, je opět symbol  $\$$ . Rozkladová tabulka slouží k určení toho, jaké přepisovací pravidlo se použije na základě symbolu na vstupu. To vše propojuje řídicí program, který čte data ze vstupu, pracuje se zásobníkem a dívá se do rozkladové tabulky a na základě všech těchto informací v každém stavu při čtení symbolu ze vstupu rozhoduje co bude probíhat dále [33].

### Rozkladová tabulka

Ještě předtím, než bude vysvětlen celý princip nerekurzivního prediktivního přístupu, je potřeba si vysvětlit, co to je a jak se vytváří rozkladová tabulka.

Rozkladová tabulka obsahuje jak terminální tak i neterminální symboly gramatiky. Řádky označují neterminální a terminální symboly a sloupce pouze terminální symboly. Po vytvoření rozkladové tabulky je parser schopen rozhodnout jaké přepisovací pravidlo použít pro odvození vstupu. Tabulka proto obsahuje pár předdefinovaných klíčových hodnot. Formálně je rozkladová tabulka zobrazení [33]:

$$M : (\Sigma \cup V \cup \{\#\}) \times (\Sigma \cup \{\$\}) \rightarrow \{expand_1, expand_2, \dots, expand_n, pop, accept, error\} \quad (15)$$

kde:

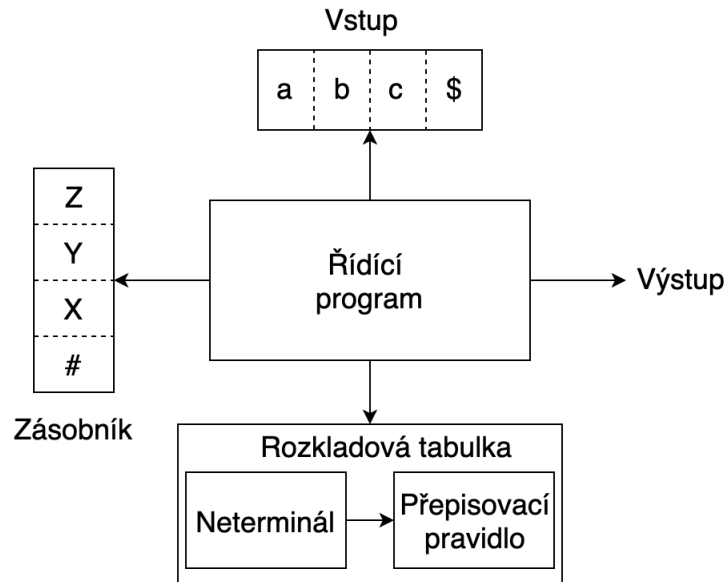
- $expand_i$  - Jestliže  $r_i : A \rightarrow \alpha$  i-té pravidlo gramatiky, na vrcholu zásobníku je neterminál  $A$ , na vstupu symbol  $a$  a  $M[A, a] = expand_i$ , potom se provede nahrazení neterminálního symbolu  $A$  na vrcholu zásobníku pravou stranou  $\alpha$  pravidla  $r_i$ .
- $pop$  - Jestliže je na vstupu terminální symbol a stejný symbol je i na vrcholu zásobníku, tak se symbol odstraní ze vstupu i ze zásobníku.
- $accept$  - značí přijetí vstupního řetězce, vstupní řetězec odpovídá gramatice.
- $error$  - značí nepřijetí vstupního řetězce, vstupní řetězec neodpovídá gramatice.

### Algoritmus

Postup pro vytvoření výše popsané rozkladové tabulky je následující [33]:

1. Všechny hodnoty v tabulce se na začátku nastaví na  $error$ .
2. Pro všechna pravidla ve tvaru  $r_i : A \rightarrow \alpha$  se provedou akce:
  - (a) Pro každý terminální symbol  $a \in FIRST(\alpha)$  se nastaví  $M[A, a] = expand_i$ .
  - (b) Jestliže  $\epsilon \in FIRST(\alpha)$ , tak se nastaví  $M[A, b] = expand_i$ , kde  $b$  jsou všechny terminální symboly z množiny  $FOLLOW(A)$ .

- (c) Jestliže  $\epsilon \in FIRST(\alpha)$  a zároveň  $\$ \in FOLLOW(A)$ , tak se nastaví  $M[A, \$] = expand_i$ .
3. Pro každý terminální symbol  $a$  se nastaví  $M[a, a] = pop$
4. Pro ukončení čtení se nastaví  $M[\#, \$] = pop$



Obrázek 14: Struktura nerekurzivního prediktivního parseru

### Princip nerekurzivního přístupu

Řídící program se rozhoduje podle symbolu na vrcholu zásobníku  $X$  a aktuálního symbolu na vstupu  $a$ . Rozhodování probíhá v následujících krocích [33]:

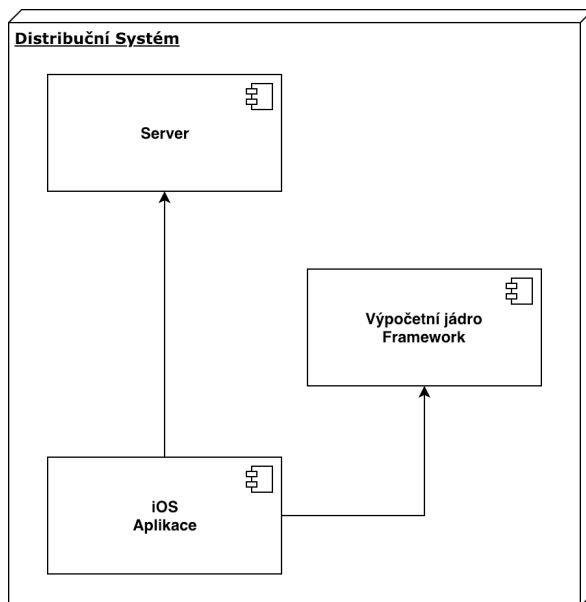
1. Jestliže  $X = \#$  a zároveň  $a = \$$ , tak potom parser úspěšně skončil.
2. Jestliže  $X = a$  a zároveň  $a \neq \$$ , tak se odstraní  $X$  ze zásobníku a přečte se další symbol ze vstupu.
3. Jestliže  $X$  je neterminální symbol, tak se provede expanze odpovídající pravé straně přepisovacích pravidel neterminálu, nebo se ohlásí error.
  - (a) Jestliže  $M[X, a]$  obsahuje pravidlo gramatiky, tak se provede expanze.
  - (b) Jestliže  $M[X, a]$  obsahuje error, tak se skončí s chybou a dále se nepokračuje. Jedná se o syntaktickou chybu ve vstupu.
4. Všechny ostatní případy se akceptují jako chybový stav, respektive syntaktická chyba ve vstupu a parser ukončuje svůj běh neúspěchem.

## 4 Analýza řešení

Cílem práce je návrh a implementace systému pro mobilní platformu (Obrázek 15). Řešení se bude skládat ze tří hlavních komponent. První komponenta je iOS aplikace, druhá komponenta výpočetní jádro a poslední komponenta je serverový backend.

Výsledný systém by měl být, co nejvíce univerzální na úrovni komponent, jednoduše rozšiřitelný o možnost přidání nové funkcionality a taktéž dobře udržitelný. Bude kladen důraz na celkovou architekturu a stejně tak na architekturu uvnitř jednotlivých komponent. Hlavní cíl je vytvořit základ systému, do kterého bude velmi snadné přidat novou funkcionality.

1. **iOS aplikace** obsahuje uživatelské rozhraní, se kterým uživatel komunikuje, jak ze strany vstupů, tak ze strany požadovaných výstupů, podle uživatelských vstupů. Mimoto je zde obsažena mezivrstva, která konvertuje vstupy a výstupy (data a datové struktury) mezi výpočetním jádrem.
2. **Výpočetní jádro** je knihovna, taktéž označované jako framework na Apple platformách, která obsahuje veškerou výpočetní logiku, algoritmy, struktury a modely používané pro zpracování požadovaných akcí. Výsledky jsou následně použité a zobrazeny uživateli v iOS aplikaci.
3. **Serverový backend** slouží hlavně ke správě uživatelů a persistenci dat. Backend poskytuje své služby skrze REST API, které dále využívá iOS aplikace.



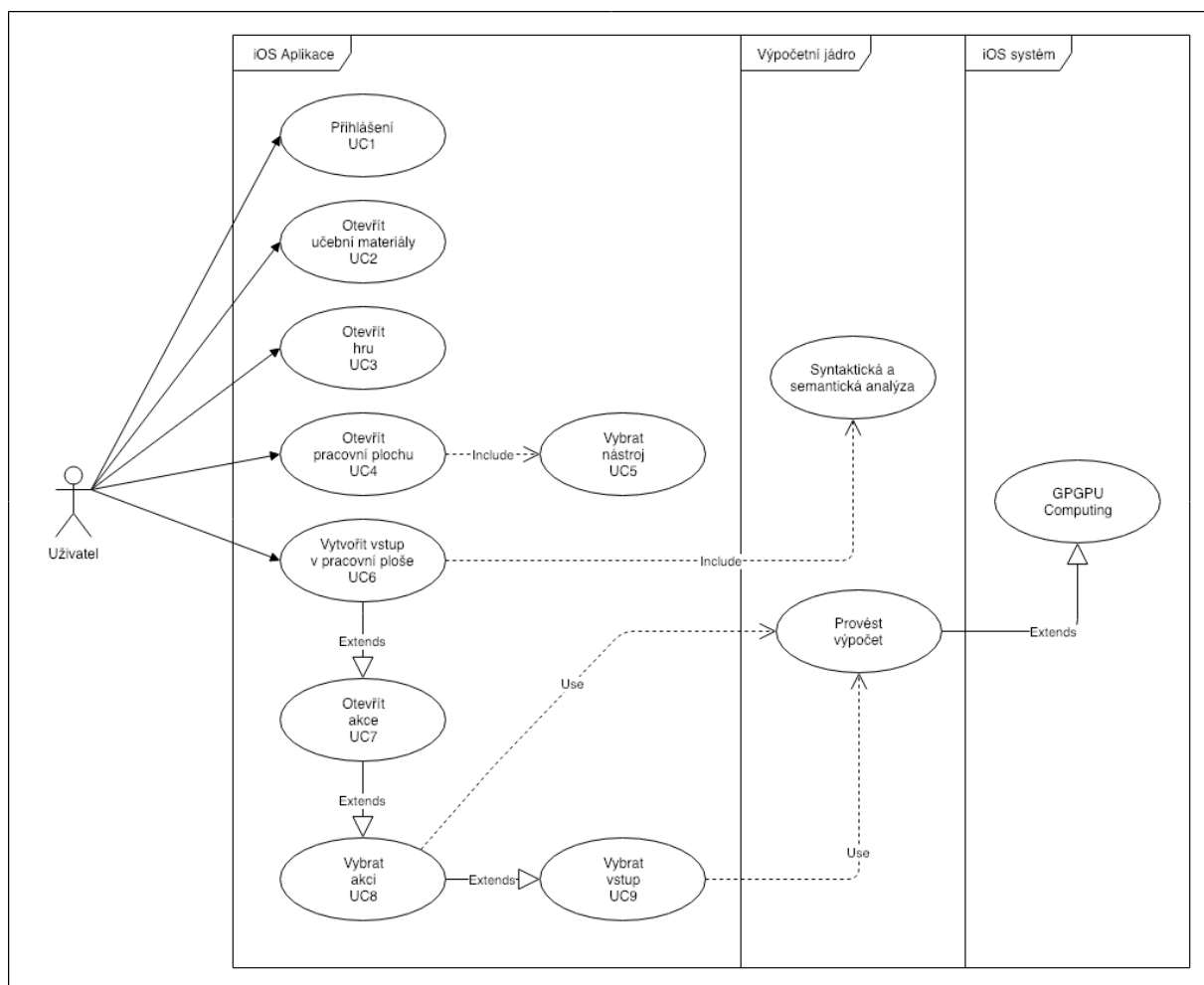
Obrázek 15: Distribuční systém - základní komponenty

## 4.1 Případy užití

Pro prezentaci případů užití je použit UML diagram [30]. Z důvodů obsáhlosti celého systému, je zde rozepsán pouze jeden hlavní případ užití z iOS aplikace, který popisuje obecný postup, jak pracovat s hlavní částí aplikace. Zde popsany případy užití (Tabulka 2 a 3) popisuje více případu užití najednou, jelikož se jedná o popis většího celku ovládání aplikace.

### 4.1.1 Obecné případy užití

Jsou zde zobrazeny obecné případy užití navrhované iOS aplikace v UML diagramu. Diagram obsahuje pouze základní a zajímavé případy užití (Obrázek 16). Z důvodu obsáhlosti systému není možné popsat všechny případy užití.



Obrázek 16: Obecné případy užití iOS aplikace



#### 4.1.2 Příklad užití - UC0

Název	Pracovní plocha
ID	UC0
Popis	Provede se akce na základě vybraného nástroje, akce v rámci nástroje a uživatelského vstupu. Výstup je uživateli prezentován do vybrané pracovní plochy a nebo se otevře nová podle typu výstupů.
Aktéři	Uživatel
Parametry	Vybraný nástroj pro práci. Vybraná akce v rámci nástroje. Uživatelský vstup v daném nástroji
Vstupní podmínky	Uživatel je přihlášen. Uživatel vybral nástroj se kterým chce pracovat
Výstupní podmínky	Uživatel má zobrazený výsledek akce

Tabulka 2: UC0 - Pracovní plocha - popis

## UC0 - Pracovní plocha

Hlavní tok		
Krok	Role	Akce
1	Uživatel	Uživatel zvolí nástroj se kterým chce pracovat
2	Aplikace	Aplikace zobrazí pracovní plochu pro vybraný nástroj
3	Uživatel	Uživatel vytvoří vstup na pracovní ploše pomocí dostupných prvků, které poskytuje vybraný nástroj, respektive pracovní plocha nástroje.
4	Uživatel	Uživatel zvolí výběr akce
5	Aplikace	Aplikace načte vstupy z pracovní plochy
6	Aplikace	Aplikace provede syntaktickou a sémantickou analýzu vytvořených vstupů.
7	Aplikace	Aplikace zobrazí seznam dostupných akcí podle vybraného nástroje
8	Uživatel	Uživatel zvolí akci
9	Aplikace	Aplikace provede výpočet na základě vybrané akce a ze vstupu z pracovní plochy
10	Aplikace	Aplikace zobrazí výsledek do pracovní plochy
Vedlejší tok		
Krok	Role	Akce
6a	Aplikace	Jestli syntaktická a sémantická analýza selhala, zobrazí se uživateli dialog s oznámením o nalezených chybách.
6a1	Uživatel	Uživatel opraví chybné vstupy a potom se přeskakuje na krok 4
8a	Aplikace	Jestli uživatel vytvořil více vstupů na pracovní ploše, aplikace zobrazí uživateli možnost výběru konkrétního vstupu
8a1	Uživatel	Uživatel vybere vstup a potom se přeskakuje na krok 9

Tabulka 3: Příklad užití UC0 - Pracovní plocha

## 4.2 Požadavky

Z analýzy dostupných řešení a požadavků na výsledný nový systém, jsou sestaveny následující funkční a nefunkční požadavky. Z důvodu obsáhlosti celého systému jsou vypsány pouze důležité a zajímavé požadavky.

### 4.2.1 Funkční požadavky

#### iOS Aplikace

##### 1. Pracovní plocha

**Priorita:** Hlavní

**Popis:** Hlavní část aplikace. Uživatel zde zadává vstupy a vybírá akce, které aplikace zpracuje ze vstupů a vygeneruje výstup do pracovní plochy.

## 2. Jednoduché ovládání

**Priorita:** Hlavní

**Popis:** V celé aplikaci musí být dodržené jednoduché ovládání.

## 3. Offline režim

**Priorita:** Hlavní

**Popis:** Aplikace je schopna provádět všechny výpočty na zařízení a není vyžadováno internetové připojení.

## 4. Učební materiály

**Priorita:** Vedlejší

**Popis:** Aplikace musí obsahovat učební materiály.

## 5. Obnovování již naučených znalostí

**Priorita:** Vedlejší

**Popis:** Aplikace nutí uživatele udržovat naučené znalosti.

## 6. Zobrazení postupů

**Priorita:** Vedlejší

**Popis:** Zobrazení postupů výpočtů jednotlivých akcí.

## 7. Statistiky

**Priorita:** Vedlejší

**Popis:** Aplikace zobrazuje uživateli souhrnný přehled o všech naučených znalostích a stavu aplikace (například čas strávený v určitých sekcích).

# Výpočetní jádro

## 1. Architektura

**Priorita:** Hlavní

**Popis:** Výpočetní jádro bude dodržovat správné programátorské praktiky a návrhové vzory.

## 2. Rozhraní

**Priorita:** Hlavní

**Popis:** Výpočetní jádro bude poskytovat přehledné API, přes které se bude s ním komunikovat.

## 3. Nezávislost

**Priorita:** Vedlejší

**Popis:** Výpočetní jádro bude obsahovat nejméně závislostí na ostatních knihovnách. Bude využívat čistý programovací jazyk Swift a jeho základní knihovny.

## Serverový backend

### 1. Správa uživatelů

**Priorita:** Hlavní

**Popis:** Slouží ke správě uživatelů.

### 2. Zprostředkování dat přes REST API

**Priorita:** Hlavní

**Popis:** Backend poskytuje data a služby prostřednictvím REST API.

### 3. Správa dat a metadat

**Priorita:** Vedlejší

**Popis:** Podpora uložení obecných dat pro běh aplikace a metadat uživatelů.

## 4.2.2 Nefunkční požadavky

### 1. iOS aplikace vyžaduje minimální verzi 10.0

**Priorita:** Vedlejší

**Popis:** Minimální podporována verze iOS systému.

### 2. iOS aplikace dodržuje standardy dané platformy

**Priorita:** Hlavní

**Popis:** iOS platforma má specifické chování a standardy, které je potřeba dodržet.

### 3. Aplikace vyžaduje internetové připojení

**Priorita:** Vedlejší

**Popis:** Pro kompletní běh aplikace je vyžadováno internetové připojení. Nicméně, hlavní části aplikace budou dostupné i bez internetového připojení.

### 4. Serverový backend

**Priorita:** Hlavní

**Popis:** Celý backendový systém je postaven na Django frameworku.

## 5 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní probíhá přes vytvoření tzv. wireframes, neboli drátěný model. Drátěný model pomáhá najít nedostatky v uživatelském rozhraní i chování aplikace. Vytvoření drátěného modelu je v dnešní době velmi rychlé, jelikož existuje spousta software, které usnadňují vytvoření požadovaného modelu. Takový software obsahuje předdefinované grafické komponenty, které dodržují standardy platformy, pro který je drátěný model vytvářen. Takový nástroj, který je více komplexní, obsahuje dokonce i možnost dynamického drátěného modelu. Například: animace, dynamický obsah, flow-mapu a nebo přechody mezi obrazovkami v mobilní aplikaci.

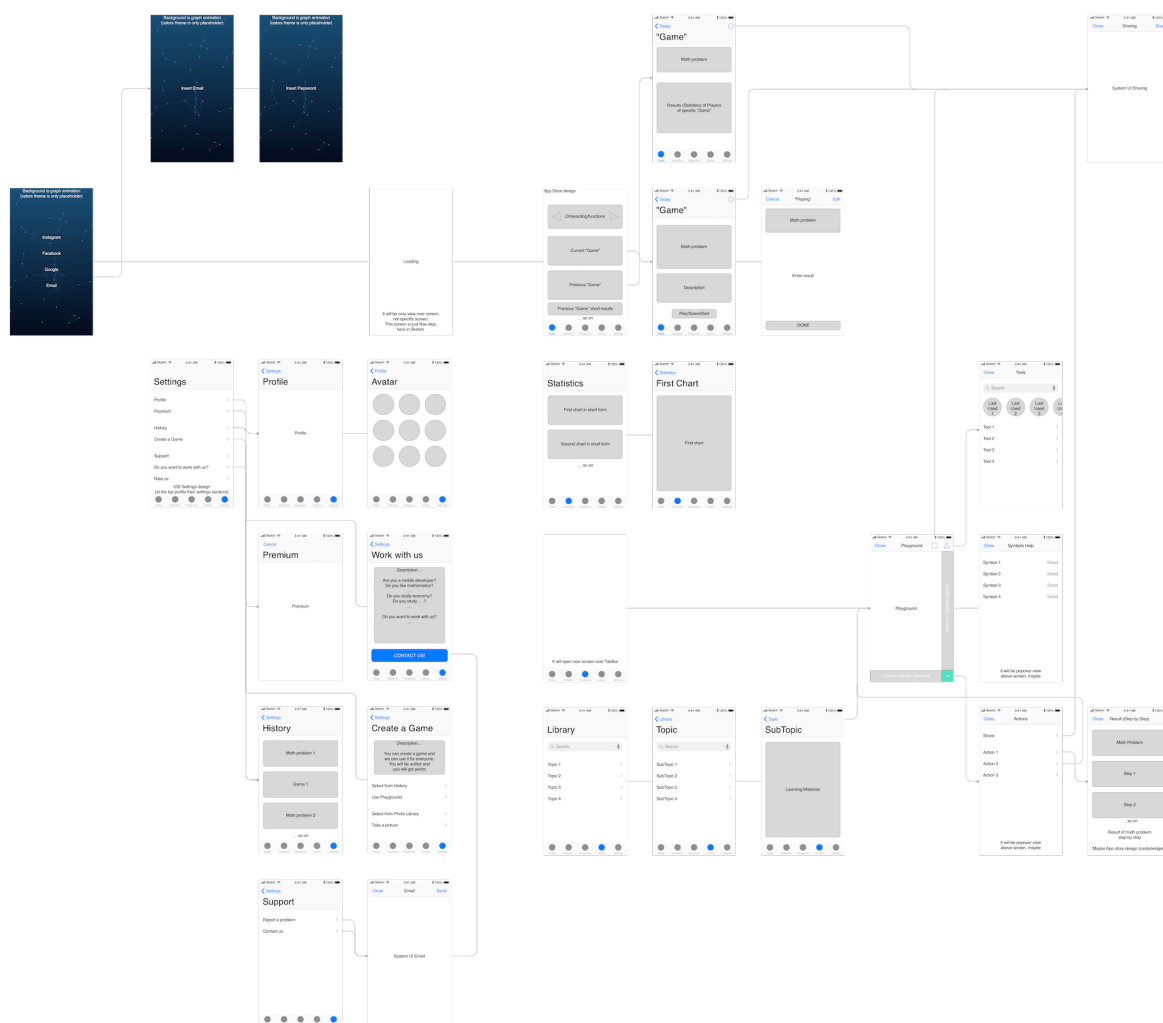
Představovaný návrh systému je vytvořen pomocí software Sketch [1]. Sketch je univerzální grafický program pro navrhování webových, mobilních a deskopových designů. Obsahuje předdefinované nástroje, grafické komponenty a ostatní nástroje pro usnadnění práce. V návrhu uživatelského rozhraní budou rozepsány pouze hlavní obrazovky iOS aplikace.

### 5.1 Flow mapa

Flow mapa zobrazuje všechny obrazovky iOS aplikace pospojované logicky mezi sebou tak, jak má uživatel možnost interagovat s aplikací (Obrázek 17).

### 5.2 Hlavní navigace

Základní navigace aplikace je řešená přes systémový tzv. TabBar. Tento element obsahuje na dně aplikace jednotlivé záložky, na které uživatel může klikat, a tím se dostávat do jiných částí aplikace. Konkrétně se zde nachází pět záložek. Všechny záložky a jejich hlavní či zajímavé vnořené obrazovky budou dále rozebrány.



Obrázek 17: Návrh - Flow mapa iOS

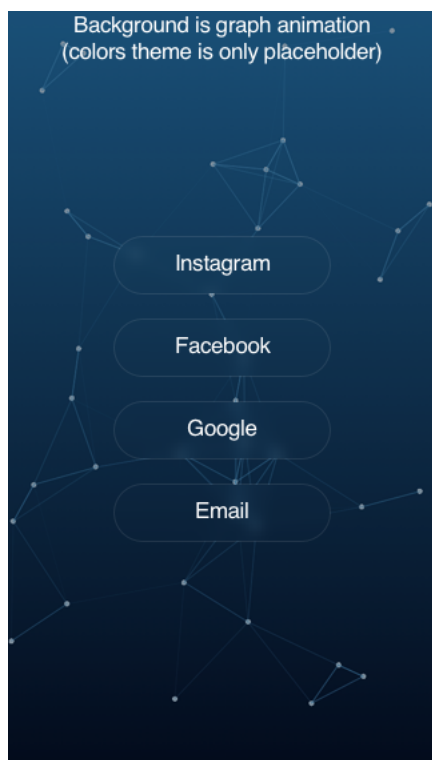
### 5.3 Obrazovka - Login

Obrazovka Login je první, která se uživateli zobrazí, při prvním spuštění aplikace. Nacházejí se zde pouze tlačítka pro různé způsoby přihlášení nebo registrace. Typy přihlášení jdou rozdělit na dvě hlavní skupiny. První typ je klasické přihlášení přes email a heslo. Druhý typ jsou sociální sítě. Při použití sociálních sítí si uživatel nemusí pamatovat přístupové údaje pro tento systém. Obrazovka se po úspěšném přihlášení (registraci) už nikdy neotevřít, uživatel zůstává přihlášený (Obrázek 18).

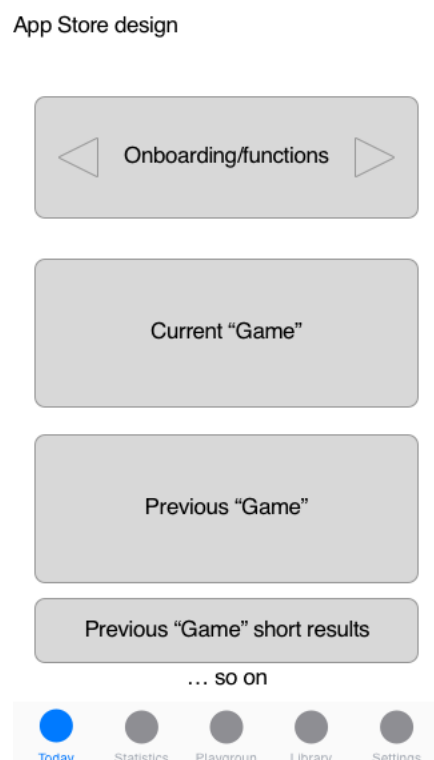
### 5.4 Obrazovka - Today

Toto je první hlavní obrazovka, kterou uživatel vidí, když je úspěšně přihlášený. Jak již anglický název obrazovky napovídá, tak slouží k zobrazení obsahu, který je aktuální v době otevření. Uživatel tak zde vidí novinky a hry, které jsou zrovna v běhu a výsledky předchozích her (bude

vysvětleno dále v textu). Další část obsahu, který je obrazovka schopna zobrazit jsou vlastnosti a schopnosti, které aplikace uživateli nabízí, jako například: pokryté partie matematiky a teoretické informatiky, které aplikace je schopna zpracovávat. Mimo těchto věcí se zde mohou nacházet různé informativní bloky, které se týkají obecně vědy, například: *Byl vytvořen nový algoritmus pro klasifikaci dat, pro více informací klikněte zde*. Hlavní obsah obrazovky je chronologicky řazen a uživatel má tak možnost procházet obsahem v čase. Obsah je zobrazen od přítomnosti až do minulosti (Obrázek 19).



Obrázek 18: Obrazovka - Login



Obrázek 19: Obrazovka - Today

## 5.5 Obrazovka - Game

Obrazovka Game, neboli Hra, slouží k zobrazení obsahu respektive funkcionalitě aplikace, která má pomoci uživateli obnovovat své dosavadní znalosti. Nicméně to může taktéž sloužit pouze pro zábavu či rozšíření obzorů.

Každá hra se skládá z definovaného problému (například: matematický problém) a popisu. První část, definice problémů, je samotná problém, který uživatel musí vyřešit a následně na něho odpovědět. Popis slouží k upřesnění zadaného problémů, tipů, nebo určení zaměření, kterých se problém týká (například: matematická analýza). Díky tomu, má uživatel možnost ještě před spuštěním hry si nastudovat nebo připomenout potřebné znalosti (Obrázek 20).

Hra může být dvojího typu:

1. Automaticky generované typy problémů
  - Nekonečná množina problémů
2. Předem definované typy problémů
  - Konečná množina problémů

Předem definované problémy jsou časově omezené a platné pouze pro určitý časový úsek. Tyto problémy jsou vytvořeny manuálně a zobrazeny všem uživatelům. Díky tomu, je možnost porovnávat uživatele mezi sebou a sestavit žebříček pro danou hru, kdo zadaný problém vyřešil správně a nejrychleji. Pomocí těchto dat je taktéž možnost sestavit globální žebříček, který zobrazuje nejlepší hráče za celou dobu existence.

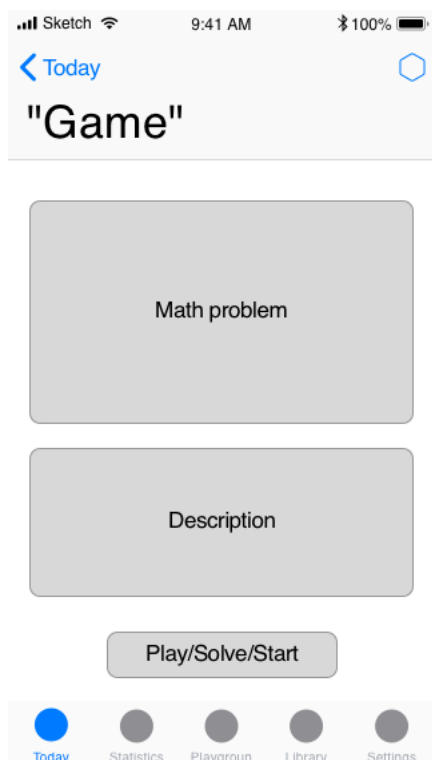
Platné časové úseky (intervaly) pro předem definovanou hru, jsou:

1. Denní
2. Týdenní
3. Měsíční

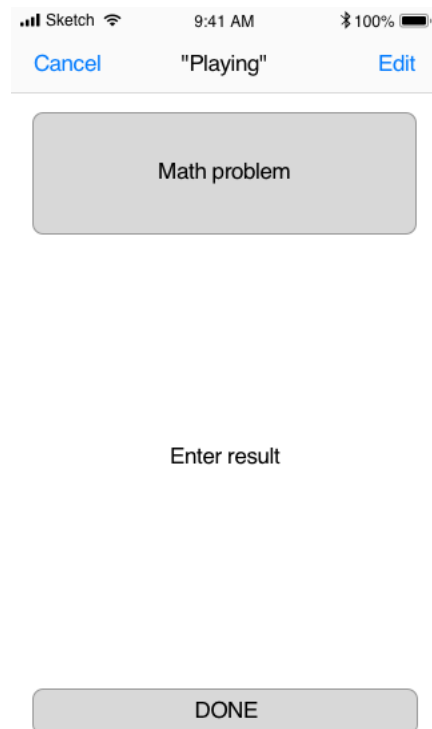
Po spuštění hry se uživateli zobrazí obrazovka Game Playing (Obrázek 21). Od této chvíle se začíná měřit čas, jak dlouho uživatel řeší zadaný problém. Uživatel nemá možnost opustit obrazovku a vrátit se zpět. Opuštění obrazovky je možné dvěma způsoby. První způsob je zadání výsledku a tím se hra ukončí. Druhý způsob je zrušit hru s tím, že už nebude možné jí opětovně hrát.

Po úspěšném dokončení hry je uživateli oznámeno, že došlo k zaznamenání jeho odpovědi. Podle typu hry (automaticky generované, nebo předdefinované problémy) je zobrazen výsledek. Jestli se jedná o předem definovaný problém, tak uživatel musí čekat na skončení časového úseku a teprve potom je zobrazen jeho výsledek (uspěl nebo neuspěl), správné řešení, postup řešení a žebříček všech úspěšných zúčastněných uživatelů seřazených podle času, který potřebovali k vyřešení problému.





Obrázek 20: Obrazovka - Game



Obrázek 21: Obrazovka - Game, hraní

## 5.6 Obrazovka - Playground

Toto je nejdůležitější obrazovka aplikace. Playground (dále v textu česky označované jako pracovní plocha) slouží k zadávání uživatelských vstupů, výběr nástrojů (matice, gramatika, regulární jazyky, atp.), zobrazení výsledků výpočtů a postupy řešení pro různé akce prováděné na základě uživatelských vstupů. Pracovní plocha otevírá další důležité obrazovky pro celkový správný chod a fungování samotných výpočtů a nástrojů. Tyto obrazovky budou dále rozepsány postupně, tak jak uživatel přichází s nimi do kontaktu při interakci s aplikací.

Jak už bylo řečeno, pracovní plocha je nejdůležitější obrazovka. Pracovní plocha je navržena tak, aby byla schopna zpracovat a zobrazit ovládací panely pro každý nástroj (matice, konečné automaty, atd.) stejně, ale zároveň dodržet přehlednost a jednoduchost uživatelského rozhraní. Tím je myšleno to, aby se pro každý nástroj zobrazily jiné ovládací prvky, ale ve stejném rozvržení. Jak bylo zmíněno v analýze současného stavu, tak mnoho aplikací zobrazilo speciální klávesnici, kde byly všechny symboly, znaky a jiné elementy pro všechny typy vstupů, tím se velmi zhoršila použitelnost a přehlednost aplikace.

Zde byl problém vyřešen následovně. Pracovní plocha se skládá ze čtyř hlavních částí (Obrázek 22):

1. Plocha pro práci se vstupem.
2. Spodní panel pro vytvoření a editaci vstupu.

3. Pravý boční panel pro vytvoření a editaci vstupu.
4. Pravé spodní hlavní tlačítko pro spuštění akcí.

Jak lze vidět z popisu, jedná se o univerzální rozvržení, kde se pouze mění obsah výše čtyř zmíněných elementů, podle toho jaký nástroj si uživatel zvolí. Po kliku na prvky ve spodním a bočním panelu dochází k vytvoření, nebo editaci vstupu na pracovní ploše. V pracovní ploše může uživatel přemísťovat vytvořené vstupy a označovat jednotlivé detaily, pro zpětnou editaci. Podle označeného prvku ve vstupu se opět přizpůsobí zobrazení obsahu ve spodním a bočním panelu tak, aby prvky v panelech odpovídali editaci daného vstupu a vybraného prvku uvnitř vstupu.

#### 5.6.1 Obrazovka - Tools

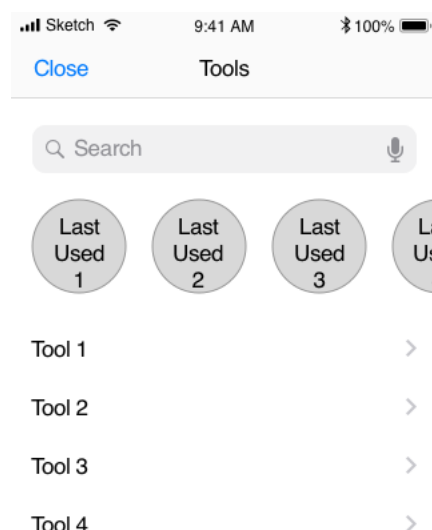
Obrazovka Tools (Obrázek 23), neboli nástroje, byly už zmíněny v předchozím textu, takže už i nastal čas definovat, co je tím přesně myšleno. Ještě předtím, než je uživateli zobrazena pracovní plocha, musí si vybrat jeden z dostupných nástrojů. Nástroj určuje, co bude na pracovní ploše zobrazeno. Nástroj se tedy vždy zaměřuje na řešení specifických problémů. Tím se myslí například nástroj pro práci s: maticemi, aritmetikou, integrály, limity, bezkontextové gramatiky, konečné automaty, nebo různé specifické nástroje pro práci s algoritmy ze strojového učení, či úplně z jiných vědeckých disciplín.

Díky tomu, jak je navrhnutá pracovní plocha a nástroje, je velmi snadné přidat nový nástroj do již existujícího systému (aplikace) a tím rozšířit funkcionalitu systému. Systém je díky tomu univerzální a přidání nového nástroje neovlivní již předchozí funkcionalitu, či existující nástroje. Zároveň je dodržena podmínka přehlednosti a jednoduchost uživatelského rozhraní, jelikož každý nástroj se stará o zpracování pouze specifických problémů. Příkladem může být to, že když uživatel pracuje s maticemi, tak nepotřebuje vidět prvky pro zadání vstupů pro končené automaty. To platí samozřejmě i naopak.

Každý nástroj je zodpovědný za konfiguraci pracovní plochy a definici již předchozích čtyřech zmíněných hlavních částí.



Obrázek 22: Obrazovka - Playground



Obrázek 23: Obrazovka - Tools

### 5.6.2 Obrazovka - Actions

Jakmile uživatel už vytvořil vstup na pracovní ploše, tak může s ním provádět dostupné akce. K tomu slouží obrazovka Actions, neboli akce (Obrázek 24). Akce jsou opět závislé na vybraném nástroji. Nástroj definuje, jaké akce bude moci uživatel spouštět na základě jeho vstupu z pracovní plochy. Pro upřesnění se jedná například o vytvoření regulárního výrazu na pracovní ploše a následně je dostupná akce převést regulární výraz na konečný automat. U matic to může být provést LU rozklad zadané matice.

Jestliže nástroj dovoluje v pracovní ploše zadat více vstupů (více matic, více regulárních výrazů, atp.), tak po vybrání akce, která má být provedena, je uživateli zobrazena obrazovka pro výběr jednoho konkrétního vstupu, nad kterým dojde ke spuštění akce.

Výsledek provedené akce má dva možné stavy. První je zobrazení výsledku do stejné pracovní plochy a díky tomu může uživatel dále pracovat s výsledkem. Možný příklad zde je již zmíněný LU rozklad matice, kde uživatel vybere vstup z pracovní plochy, matici A, a po dokončení akce se do stejné pracovní plochy zobrazí dvě nové matice a to L a U. Druhý způsob zobrazení výsledku je otevření nové pracovní plochy podle typu výsledku. Následuje opět příklad. Při převodu regulárního výrazu na konečný automat dojde k otevření nové pracovní plochy, o kterou se stará nástroj pro správu konečných automatů. Uživatel má vždy možnost se vrátit zpět ke své původní pracovní ploše, nedochází ke ztrátě informací.

### 5.6.3 Obrazovka - Step by Step

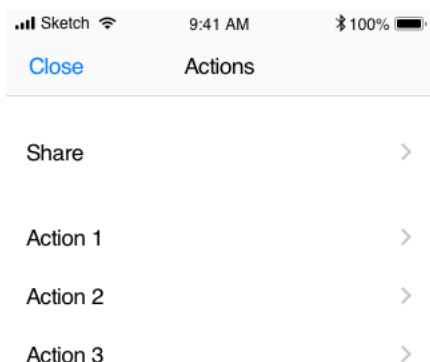
V hodně případech je důležité pro uživatele vidět postup řešení, jak se daná akce provedla. Například násobení matic. Uživatel má tak možnost si vybrat akci z obrazovky Akce, která kromě zobrazení výsledku v pracovní ploše zobrazí postup řešení. K tomu slouží obrazovka Step by Step, neboli Krok za Krokem (Obrázek 25).

Obrazovka se skládá z jednotlivých kroků, které byly prováděny z původního zadaného problému až do výsledného řešení. Kroky jsou prezentovány ve vertikální tabulce, kde nahoře (na začátku) je původní zadaný problém a na konci tabulky je výsledné řešení. Uživatel je tak schopen si procházet postup řešení krok za krokem s možností se kdykoliv podívat dopředu a nebo se vrátit zpět.

Každý krok se skládá z následujících částí:

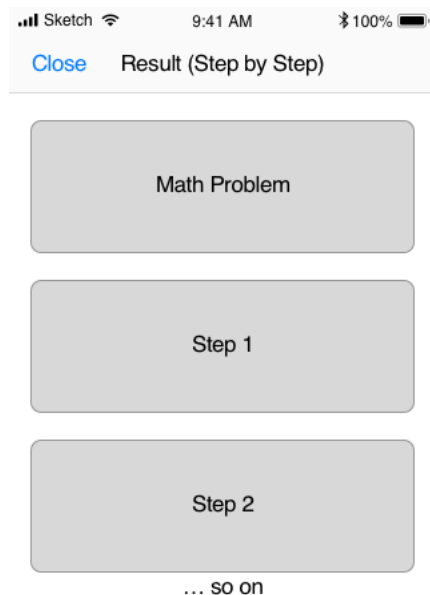
1. Název zvolené metody (či úpravy) v daném kroku.
2. Popis a vysvětlení proč došlo ke zvolení metody.
3. Je-li to možné, tak grafická prezentace vstupu po provedení metody.

Zobrazení postupu je velmi složitá věc, a musí být provedena velmi opatrně pro každou akci, aby se uživatel neztrácel v postupu řešení a zároveň kroky byly co nejvíce srozumitelné.



It will be popover view  
above screen, maybe

Obrázek 24: Obrazovka - Actions



Result of math problem  
step by step

Maybe App store design (cards/widgets)

Obrázek 25: Obrazovka - Step by Step

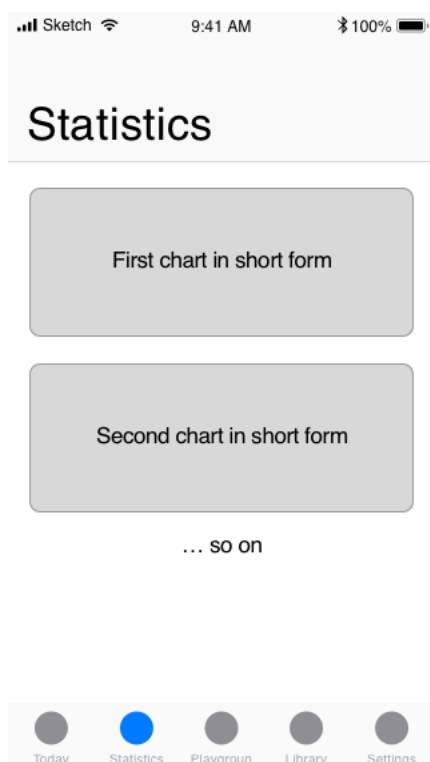
## 5.7 Obrazovka - Statistics

Tato obrazovka slouží pro zobrazení všech statistik, grafů, úspěchů a ostatních vizuálních komponent pro prezentaci dat. Data, která jsou zde zobrazována, jsou získána z uživatelského chování uvnitř aplikace. Na základě těchto dat jsou uživateli vykresleny grafy a má tak možnost si shrnout své dosavadní výsledky, a vidět v čem je jeho silná, či slabá stránka. Kromě toho by statistiky měly sloužit jako motivace uživateli k zlepšení a udržení již naučených znalostí (Obrázek 26).

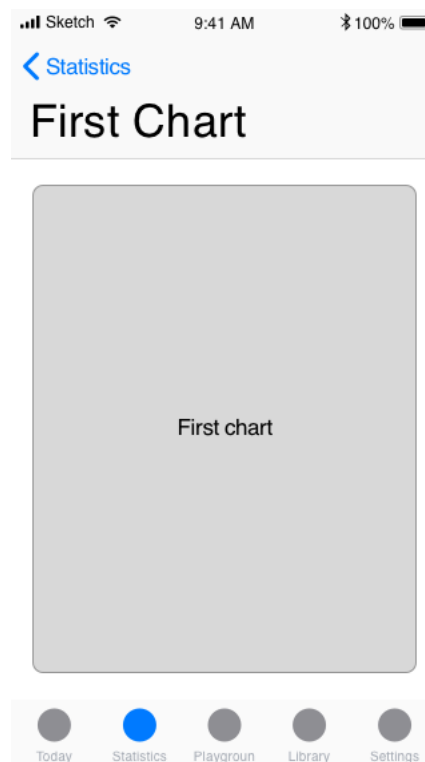
Z hlavní obrazovky, kde se nacházejí všechny grafy, má uživatel možnost si každý jednotlivý graf otevřít a vidět tak přesnější a rozsáhlejší informace pro daný kontext (Obrázek 27).

Možné měřitelné vlastnosti jsou:

1. Čas strávený v aplikaci a v různých částech aplikace.
2. Frekvence používání typu pracovní plochy a nástrojů.
3. Typ a frekvence zobrazení učebních materiálů.
4. Frekvence plnění úkolů.
5. Pravidelná aktivita.
6. Reakce na nové funkce aplikace.



Obrázek 26: Obrazovka - Statistics



Obrázek 27: Obrazovka - Statistics, detail

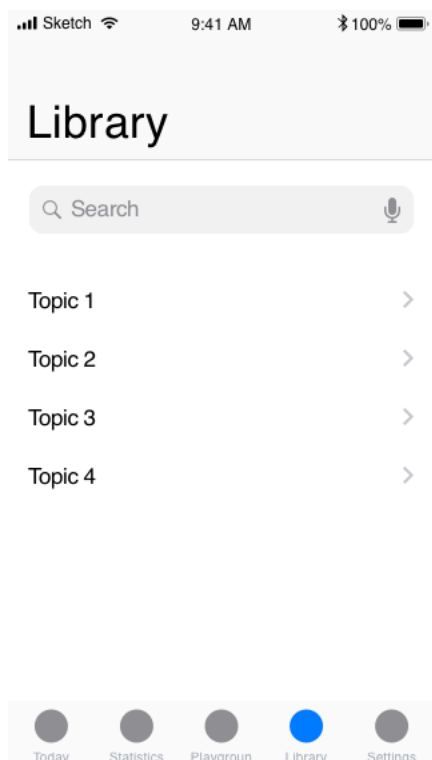
## 5.8 Obrazovka - Library

Obrazovka Library, neboli Knihovna, jak už název napovídá, slouží ke studiu a zobrazení učebních materiálů. Učební materiály jsou rozděleny podle zaměření, kterým se zabývají (Obrázek 28). Uživatel může rychle vyhledat učební materiál, nebo se postupně proklikat knihovnou. Učební materiály jsou seskupovány do logických celků, například algebra, teoretická informatika, strojové učení atd. Tyto skupiny se dále větví hlouběji, tím jak uživatel prochází knihovnou, se dostává do více zaměřených témat. Například skupina algebra, obsahuje v sobě pod skupinu matice a tato skupina obsahuje učební materiál pro násobení matic. Může taktéž obsahovat další skupinu například rozklady matic.

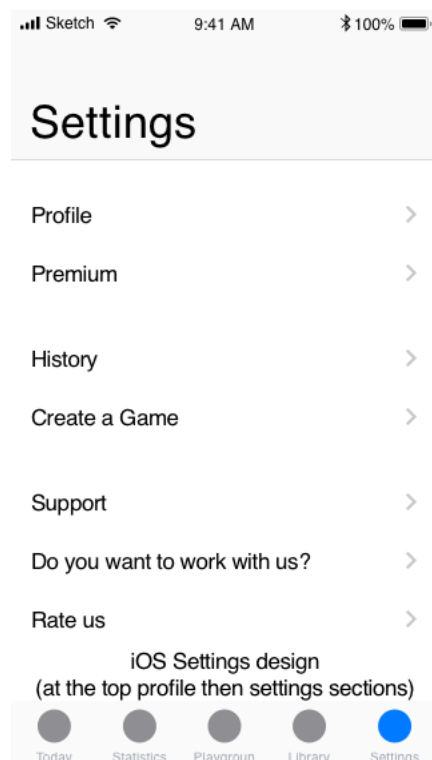
Grafické znázornění učebních materiálů je klasickým stylem ve formě textů a obrázků. Podle potřeby může být doplněn audiovizuální obsah. Po nastudování tématu, si uživatel může jít ověřit znalosti do automaticky generovaných her, kde budou zobrazeny problémy podle vybraného tématu.

## 5.9 Obrazovka - Settings

Poslední zde zmíněná obrazovka je Settings, neboli Nastavení (Obrázek 29). Toto je obecná obrazovka pro konfiguraci uživatelského profilu, ohodnocení aplikace, napsání na podporu při nalezení libovolné chyby v aplikaci, možnost zapojit se do vývoje, či vytvořit vlastní zadání pro manuální typ hry. A další jiné potřebné podpůrné vlastnosti aplikace.



Obrázek 28: Obrazovka - Library



Obrázek 29: Obrazovka - Settings

## 6 Návrh architektury

Zvolení správné architektury celého systému je jednou z hlavních částí, nad kterou je potřeba se zamyslet ještě předtím, než se začne samotným vývojem. Architektura systému ovlivňuje jaké komponenty se použijí a jak budou mezi sebou vzájemně komunikovat, či možné automatizované testování. Velmi často je potřeba dbát na to, aby zvolená architektura byla vhodná pro vytvářený systém, jelikož s tím souvisí jak následná údržba systému, tak i možné rozšíření.

Na architekturu se lze podívat ze dvou úhlů pohledu:

1. Architektura celého systému ve vztahu rozložení a komunikace jednotlivých komponent.
2. Architektura uvnitř jednotlivých komponent.

Architektura systému jako celku popisuje jak a které hlavní části systému spolu komunikují a co vše je potřebné k nasazení do ostrého provozu. Naopak architektura uvnitř každé komponenty zajišťuje správný vývoj a rozvržení jejich vlastních komponent, například architektura mobilní iOS aplikace.

### 6.1 Diagram komponent

Celý systém obsahuje pár základních hlavních komponent. Příslušné komponenty mezi sebou komunikují a zajišťují tak správný běh celého systému (Obrázek 30).

#### Server

Serverová část se stará o správu uživatelů, uložení dat, byznys logiku a zprostředkování dat a jiných podpůrných vlastností skrze REST API.

#### Interlayer

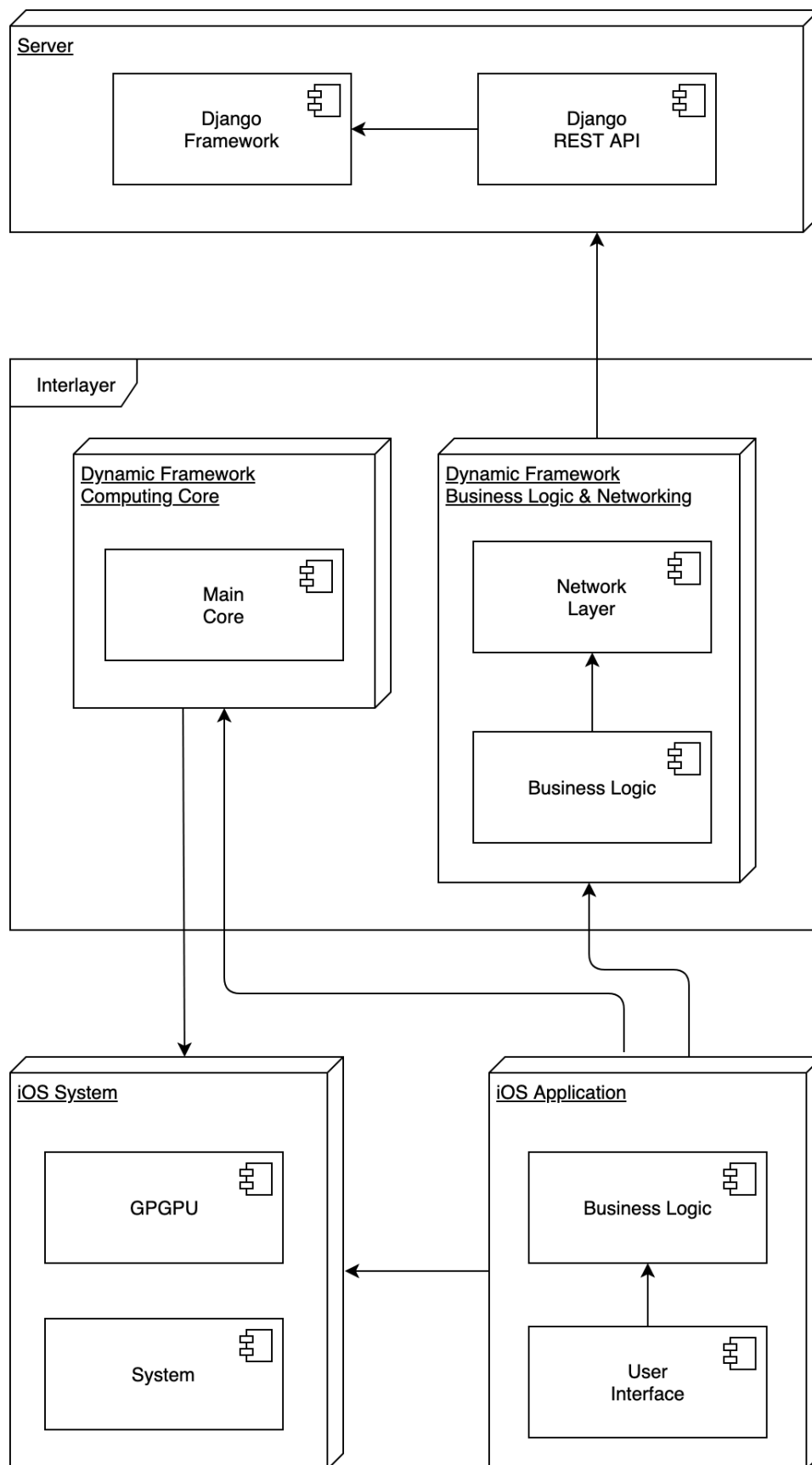
Interlayer, respektive mezivrstva, obsahuje byznys logiku, lokální databázi, komunikace se serverem a hlavně samotné výpočetní jádro. Jádro se stará o všechny matematické výpočty, datové struktury, podpůrné podprogramy a syntaktickou analýzu.

#### iOS Application

Tato komponenta je samotná iOS aplikace. Aplikace využívá mezivrstvu, respektive její výpočetní jádro, byznys logiku a networking. Samotná aplikace hlavně řeší uživatelské rozhraní pro všechny výpočetní editory.

#### iOS System

Poslední nejnižší komponenta je samotný iOS systém. Tuto komponentu využívá jak iOS aplikace, tak samotné výpočetní jádro. Pro rychlejší, efektivní a paralelní výpočty se často hodí provádět operace na grafickém jádře, které poskytuje masivní paralelizmus. Taktéž je zde možnost využít koprocessor pro strojové učení a neuronové sítě.



Obrázek 30: Diagram komponent



## 6.2 iOS

Pro nově vyvíjený systém na platformě iOS [12] je potřeba najít vhodnou architekturu. Možností se nabízí několik ať už od základní MVC architektury po MVVM, či Viper, nebo jiná podle vlastních potřeb. Každá má své chyby a přednosti. Taktéž se každá architektura hodí na jiný typ projektu. Zde jsou stanoveny hlavní požadavky, podle kterých bude vybrána finální architektura:

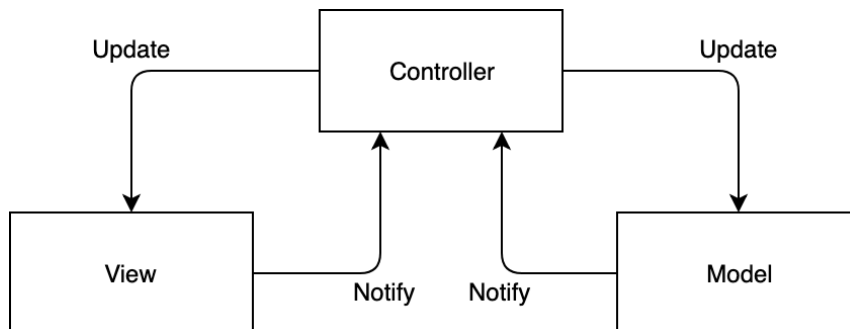
1. Přehlednost jednotlivých komponent.
2. Snadná udržitelnost.
3. Snadné přidání nové funkcionality.
4. Možnost provádět automatizované testy.
5. Zajištění minimalizace základních chyb v celém projektu.
6. Rozsáhlost a obtížnost systému.

### 6.2.1 MVC

Standardní architektura na platformě iOS je mírně upravená MVC architektura [8]. Tato architektura je velmi rozšířená a často používána. Bohužel má spoustu nedostatků. Jako hlavní nedostatek lze uvést, že není vhodná pro rozsáhlejší projekty. Při rozsáhlejších projektech se architektura stává špatně použitelnou, udržitelnou a rozšiřitelnou. Díky tomu se pak objevují zbytečné chyby při vývoji.

MVC se skládá ze tří hlavních částí (Obrázek 31). Model, View a Controller. Každá část zpracovává konkrétní část logiky, pro kterou je určena. Části, nebo bloky, mezi sebou komunikují a vyměňují si data nebo informace o změnách a událostech.

Hlavní blok je controller, který se stará o propojení, respektive komunikaci mezi view a model blokem. Model se stará o zpřístupnění dat a byznys logiku. Poslední view je blok, který má na starosti uživatelské rozhraní. Jak lze z popisu vidět, při složitých projektech se může controller stát velmi obsáhlým a složitým elementem.



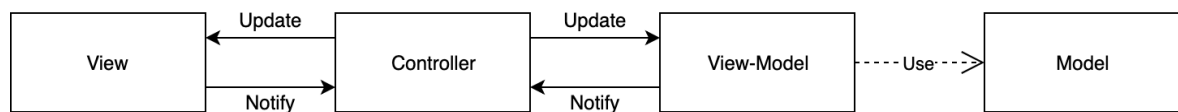
Obrázek 31: MVC architektura

### 6.2.2 MVVM

V poslední době se velmi rozšířila MVVM architektura [35]. MVVM odstraňuje spoustu nedostatků, kterými trpí MVC. Tato architektura více rozvrstvuje kód do více bloků, které jsou ve výsledku mnohem jednodušší. MVVM je velmi flexibilní a proto nabízí jednoduchou možnost, jak si upravit základní model podle svých potřeb pro daný projekt. Z toho důvodu existuje spousta verzí, které se liší pouze v drobnostech.

Zde se architektura dělí na čtyři základní bloky (Obrázek 32). Model, View, Controller a View-Model. Blok Model je stejný jako v MVC, poskytuje data a byznys logiku. View je taktéž stejné, stará se pouze o uživatelské rozhraní. Rozdíl přichází v blocích controller a view-model. View-Model je zde nový blok, který se primárně stará o komunikaci s modelem a následně předzpracování a zformátování dat získaných z modelu. Může zde taktéž být obsažena byznys logika. Díky tomu se controller blok stal velmi primitivním. V tomto případě zde hraje roli pouze prostředníka mezi view a view-model blokem. Bere data z view-model bloku a posílá je do view bloku. Naopak když uživatel provede nějakou akci, která vyžaduje nové data, nebo například komunikaci se serverem, tak je controller notifikován o příchozí akci z uživatelského rozhraní a dále předá informaci do view-model bloku.

true



Obrázek 32: MVVM architektura

Z důvodu požadovaných vlastností na nově vyvíjený systém je zvolena architektura MVVM.

### 6.2.3 Dynamický framework

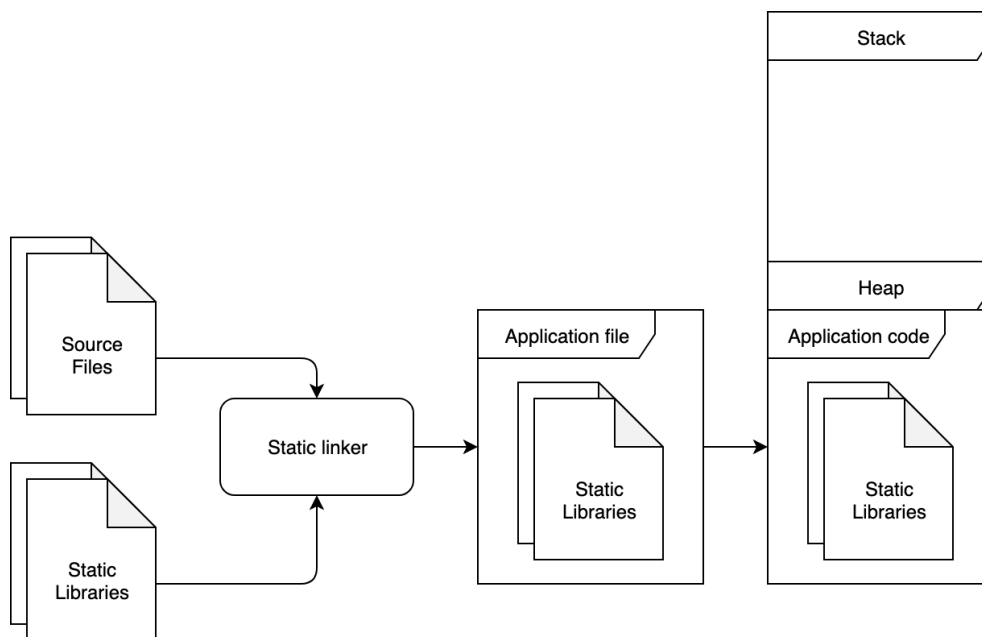
Při vývoji na Apple platformách se často používá dynamický framework [9]. Jedná se o dynamickou knihovnu. Je několik důvodů proč použít framework. Například mít možnost použít implementovanou logiku i v jiných aplikacích než pouze pro iOS, mít více rozvrstvenou architekturu, lépe udržitelné a rozšiřitelné. Výsledný framework pak lze použít například i pro macOS, watchOS nebo tvOS.

Důvod může být i technického původu. Jsou tři možnosti při vývoji. První možnost je nepoužít vůbec žádný framework, potom je kód pouze obsažen ve výsledné aplikaci a nelze ho použít jinde. Druhé dvě možnosti jsou statický framework a dynamický framework.

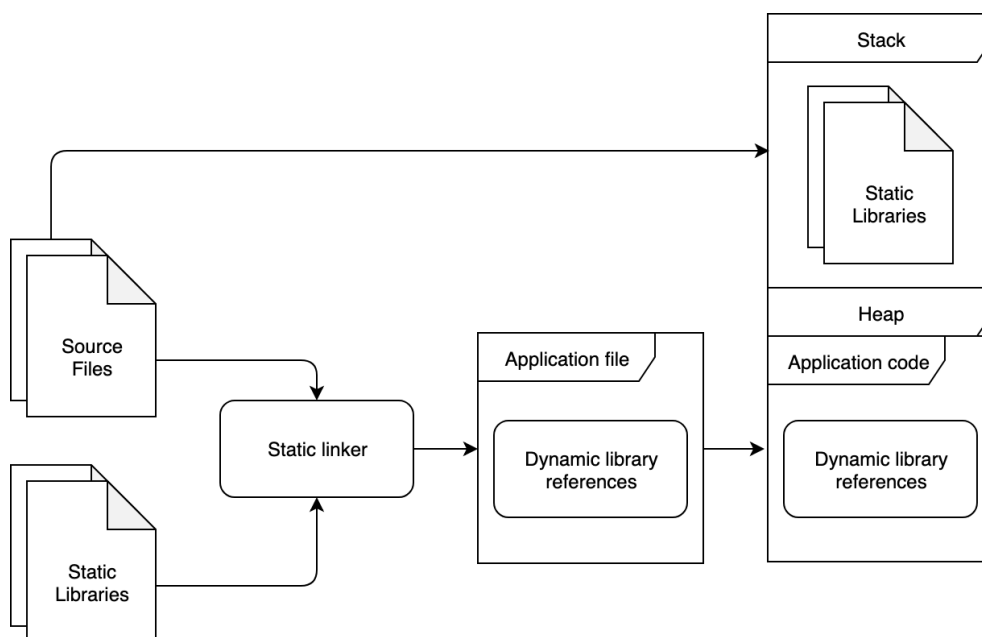
Dynamický framework (Obrázek 34) funguje odlišným způsobem, než statický (Obrázek 33). Hlavní výhody použití dynamického frameworku jsou zrychlení startu aplikace, redukce výsledné aplikace v binární podobě a redukce ploužité operační paměti, při běhu aplikace. Dynamický

framework není zahrnut při kompilaci do spustitelného souboru aplikace. Jeho načtení probíhá v momentě, až je to potřeba, respektive až aplikace potřebuje využít jeho funkci. Mimo to je framework umístěn v paměti označované jako stack a aplikace je umístěna v paměti typu heap. Kde aplikace má pouze odkaz na framework do paměti stack. Jestliže několik aplikací potřebuje využít tentýž stejný dynamický framework, tak si pouze sáhnou do paměti stack. Díky tomu nemusí být dynamický framework součástí binárky samotné aplikace.

true



Obrázek 33: Statický framework [9]



Obrázek 34: Dynamický framework [9]

## 6.3 Doménový model

Jednou z hlavních věcí, které je taky potřeba navrhnout ještě předtím, než se začne se samotným vývojem je relační databázové schéma. Ve fázi příprav se tím lze vyhnout případným problémům, které by vznikly v implementační fázi. Zde se nacházejí dvě hlavní schémata. První slouží pro iOS aplikaci a druhé pro server.

### 6.3.1 iOS

V mobilním klientovi stačí mít zjednodušené databázové schéma oproti serveru. Skrze rozdílovou synchronizaci se aktualizují data v databázi mezi aplikací a serverem (Obrázek 35).

#### User

Entita - User obsahuje základní informace o aktuálním přihlášeném uživateli.

#### Task

Entita - Task obsahuje rozpracované data v pracovní ploše. Díky tomu se uživatel může kdykoliv vrátit zpět k rozdělanému problému, kterým se zabýval.

#### TaskHistory

Entita - TaskHistory je propojovací tabulka mezi uživatelem a entitou Task, jelikož kardinalita vztahu mezi těmito entitami je  $N : M$ . Mimo to tato entita je schopna zaznamenávat historie provedené v průběhu času. Uživatel má tak možnost se vracet zpět, nebo dopředu.

#### Game

Entita - Game obsahuje informace o hře.

#### GameResult

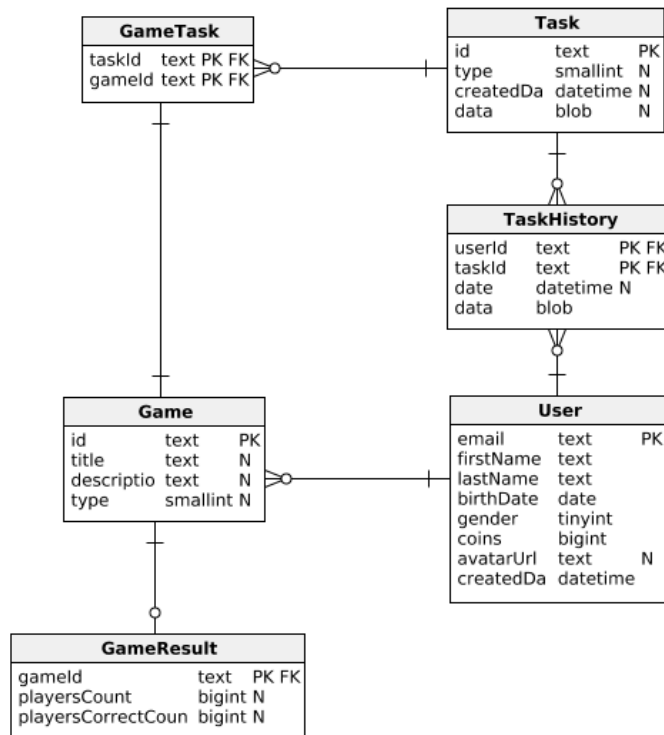
Entita - GameResult obsahuje informace o výsledcích hry.

### 6.3.2 Server

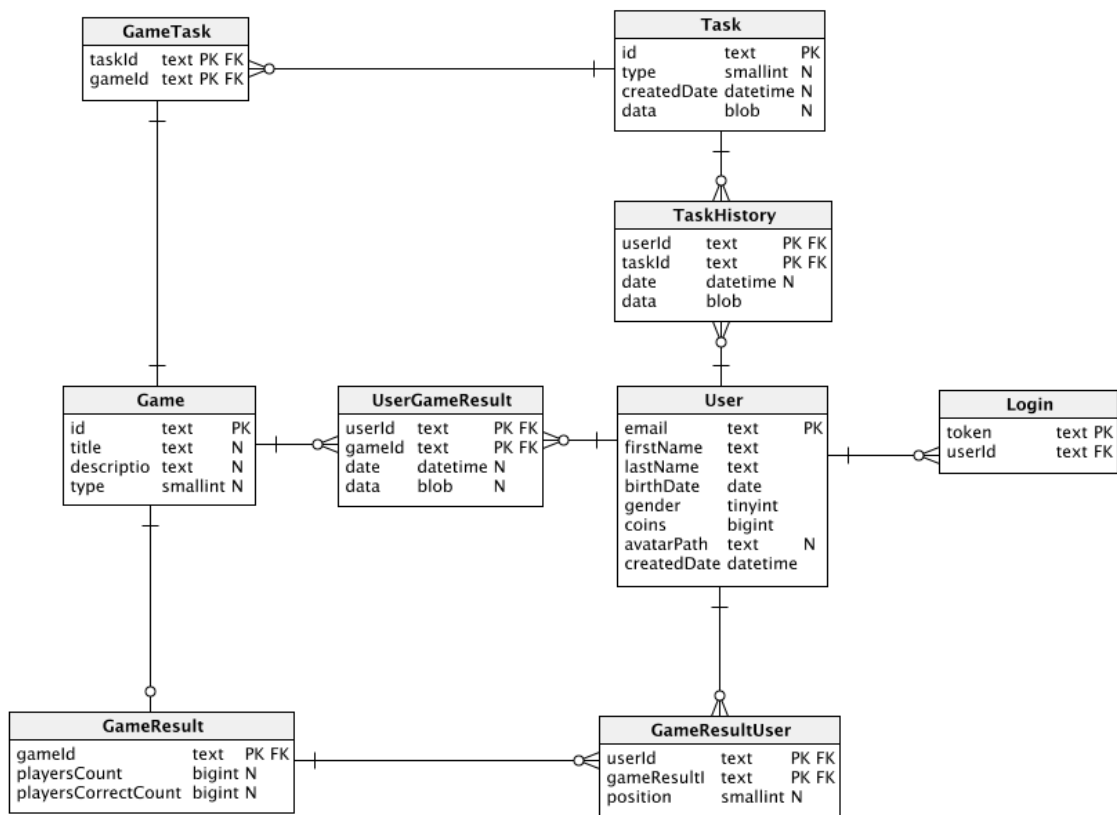
Serverové relační databázové schéma je trochu obsáhlejší než na iOS. Nicméně se jedná hlavně o propojovací tabulky, které jsou nutné z důvodu, že server ukládá informace o všech uživateliích na rozdíl od iOS aplikace, která pracuje s jedním uživatelem (Obrázek 36). Jelikož jsou zde entity stejné, kromě Login entity, tak zde nebudou popsány.

#### Login

Entita - Login obsahuje informace o přihlášeném uživateli. Každý uživatel může mít mnoho loginů. Každý login obsahuje token, se kterým aplikace komunikuje se serverem. Server pak je schopen ověřit, jestli je token platný a zároveň o jakého uživatele se jedná.



Obrázek 35: iOS databázové relační schéma



Obrázek 36: Serverové databázové relační schéma

## 7 Implementace

Při samotném vývoji se používají nástroje, které zrychlují a zkvalitňují vývoj. Stejně tak, jako knihovny třetích stran, které řeší rutinní záležitosti, které by bylo jinak potřeba vždy naprogramovat.

Pro vývoj serverové části byl použit programovací jazyk Python. Pro iOS aplikaci a dynamické frameworky byl použit programovací jazyk Swift.

### 7.1 Nástroje

Níže je uveden seznam nástrojů, které byly využity během vývoje.

#### 7.1.1 Xcode

Xcode je integrované vývojové prostředí [11]. Slouží pro vývoj softwaru pro všechny Apple platformy. Jedná se o rozsáhlé prostředí, které usnadňuje vývoj. S Xcode se také dodává řada podpůrných softwarů, jako jsou simulátory, které simulují reálné zařízení. Dále taky nástroje pro ladění a hledání chyb při běhu aplikace. Software pochází přímo od společnosti Apple.

#### 7.1.2 Carthage

Při vývoji se často používají knihovny od třetích stran, které nejsou přímo dostupné v systému pro který se vyvíjí. Je potřeba mít nástroj, který bude spravovat použité nástroje od třetích stran. Carthage je k tomu speciálně určený [16]. Stará se o aktualizaci, hlídá závislosti knihoven mezi sebou a taky správné verze jednotlivých knihoven. Funguje pro všechny Apple platformy (iOS, tvOS, macOS a watchOS). Jedná se o CLI nástroj.

#### 7.1.3 Swiftgen

Swiftgen je podpůrný CLI nástroj, který slouží ke generování Swift zdrojových kódů z obrázků, textů (viditelných v uživatelském rozhraní) a ostatních drobných vlastností použitých v aplikaci [48]. Tento nástroj pouze usnadňuje práci při vývoji a zabráňuje chybám ze strany vývojáře, aby nedošlo k překlepu názvu obrázků, či textu. Standardní cestou musí vývojáři napsat název obrázku nebo textového řetězce natvrdo do kódu. Swiftgen udělá z obrázků a textů výčtové typy, které se následně ve zdrojovém kódu používají jako konstanta.

#### 7.1.4 PyCharm

PyCharm je integrované vývojové prostředí od společnosti JetBrains, který slouží pro vývoj softwaru v programovacím jazyce Python [28]. Nástroj je rozsáhlý a poskytuje veškeré nástroje potřebné pro vývoj, podobně jako Xcode.

### 7.1.5 GIT

GIT je verzovací nástroj, respektive systém, který slouží pro správu změn v souborech při vývoji software [13]. Umožňuje vidět veškeré provedené změny v čase a možnost přepnout se do daného bodu. Podporuje i další vlastnosti jako je nelineární vývoj nebo distribuovaný vývoj.

## 7.2 Knihovny

Použité knihovny od třetích stran slouží ke zrychlení vývoje. Zde jsou použité pouze základní knihovny třetích stran a základní knihovny, které jsou přímo v iOS systému poskytované Apple.

### 7.2.1 RxSwift, RxCocoa

Jedná se o ReactiveX (Reactive extensions) [37], je to způsob programování založen asynchronním programováním se dvěma základními návrhovými vzory: pozorovatel a iterátor [37]. Základní princip je takový, že pozorovatel provede přihlášení na pozorovatelnou sekvenci. Sekvence je libovolný proud dat, či informací. Když se v sekvenci objeví data, dorazí všech pozorovatelům, kteří poslouchají danou sekvenci. Pozorovatelé pak mají možnost zpracovat a zareagovat na příchozí data. ReactiveX je v dnešní velmi populární skoro na všech platformách, jak od těch mobilních, tak přes deskopové až po web.

Na iOS byl tento způsob použití pro propojení, ve MVVM architektuře, View-Modelu a View přes Controller. Kde se na základě události v pozorovatelné sekvenci provede aktualizace dat v uživatelském rozhraní. Platí to i opačně, že z uživatelského rozhraní putuje sekvence do View-Modelu, kde se daná akce zpracuje.

### 7.2.2 Alamofire

Alamofire usnadňuje práci se sítí na straně mobilní aplikace [3]. Využívá základní iOS knihovny pro komunikaci po síti, které zaobalila a zjednodušila celkové použití. Zde to slouží pro komunikaci se serverem a výměnou dat. Jedná se o velmi rozšířenou knihovnu.

### 7.2.3 Charts

Všichni mají rádi data prezentované v pěkných grafech. Bohužel systémové knihovny na tohle neexistují. Charts je knihovna od třetí strany, která poskytuje vykreslení obrovskou škálu typů grafů [18].

### 7.2.4 CoreData

Tohle je systémová knihovna na práci s databází. Knihovna zaobaluje klasické SQLite a přidává objektový přístup a práci s databází. Jedná se o velmi robustní knihovnu. Mimo jiné poskytuje i možnost navrhnout doménový model v grafickém prostředí, které pak vygeneruje příslušné entity, respektive třídy, jako Swift zdrojový kód [6].

### 7.2.5 Metal, SpriteKit

Opět systémové knihovny, které slouží primárně k vývoji 3D a 2D her. Metal je velmi podobný OpenGL [7]. Jedná se o nízko přístupové rozhraní ke grafickému procesoru. Nicméně ne vždy je nutné použít masivní knihovnu pro 3D a proto se zde také nachází SpriteKit, který slouží pouze pro vývoj 2D her [10]. SpriteKit byl použit pro implementaci grafové animace.

### 7.2.6 Crashlytics

Jakmile je aplikace určená pro produkci a začnou ji používat reální uživatelé, tak se ať chceme nebo ne, stane, že aplikace havaruje. V tomto případě je potřeba vědět o tom, že havárie na straně uživatele nastala a identifikovat místo ve zdrojovém kódu, ve kterém došlo k havárii. K tomu účelu byla použita knihovna Crashlytics [24]. Knihovna hlídá havárie a když nastane, tak je odešle na server, kde si je může vývojář prohlédnout a analyzovat. Knihovna funguje v aplikaci bez uživatelského vědomí, není vyžadována žádná uživatelská akce.

### 7.2.7 Django

Django je webový framework používaný na straně backendu, ale i frontendu [21]. Jedná se o robustní framework, který usnadňuje práci s databází, byznys logikou a zprostředkování dat skrze API. Vývojář při malém úsilí dosáhne velmi rychlého a kvalitního výsledku, jelikož Django obsahuje spoustu předdefinovaných funkcí jako je: administrativní rozhraní, šablony, lokalizace, cachovací systém, databázi a mnoho dalšího. Pro zprostředkování dat skrze REST API slouží doplňující REST framework. Tento framework přidává spoustu dalších předdefinovaných funkcí jako je: ověření uživatelů, serializace a deserializaci dat, oprávnění, stránkování, validátory a mnohé další funkce.

## 7.3 Prvky architektury

V mnoha případech, při řešení jednotlivých podproblémů, je vhodné použít návrhový vzor. Návrhové vzory řeší konkrétní specifickou oblast, kterou popisují jak slovně, tak i UML diagramem. Je tak možné a zároveň by mělo být povinností, vždy takový vzor použít při implementaci, jelikož jsou osvědčené a není potřeba vymýšlet vlastní řešení, jinými slovy není potřeba znova vynalézat kolo. Zde budou rozepsány pouze ty nejpoužívanější vzory, které byly použité při implementaci.

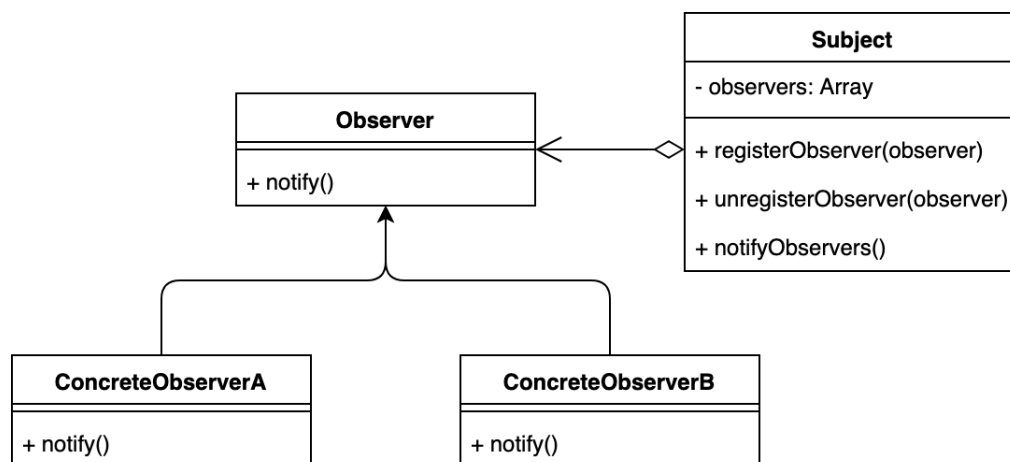
### 7.3.1 Observer

Jedná se o jeden z nejpoužívanějších návrhových vzorů (Obrázek 37). Observer popisuje obecný problém, kde je potřeba sledovat změny atributu, dat, nebo čekat na různé akce [44]. Princip je takový, že se libovolný objekt může zaregistrovat jako observer na konkrétní akci. Jakmile daná akce nastane dojde k notifikaci všech objektů, respektive observerů, kteří se předtím zaregistrovali. Následně může každý objekt provést na základě této notifikace potřebné operace.

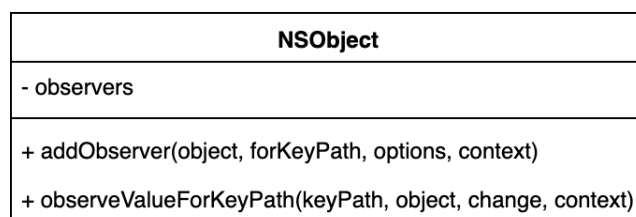


V iOS aplikace se observer vzor používá velmi často (Obrázek 38). Ať už samotná RxSwift knihovna, která je na tom postavena, tak i systémové knihovny. Každý objekt, který má předka systémovou třídu NSObject, tak může vystupovat jak v roli pozorovatele, tak i v roli poskytovatele. Mimo RxSwift (kde je využití samozřejmostí) se tento vzor využívá hlavně pro aktualizaci uživatelského rozhraní na základě nějaké akce, nebo při změně dat.

true



Obrázek 37: Obecný návrhový vzor observer



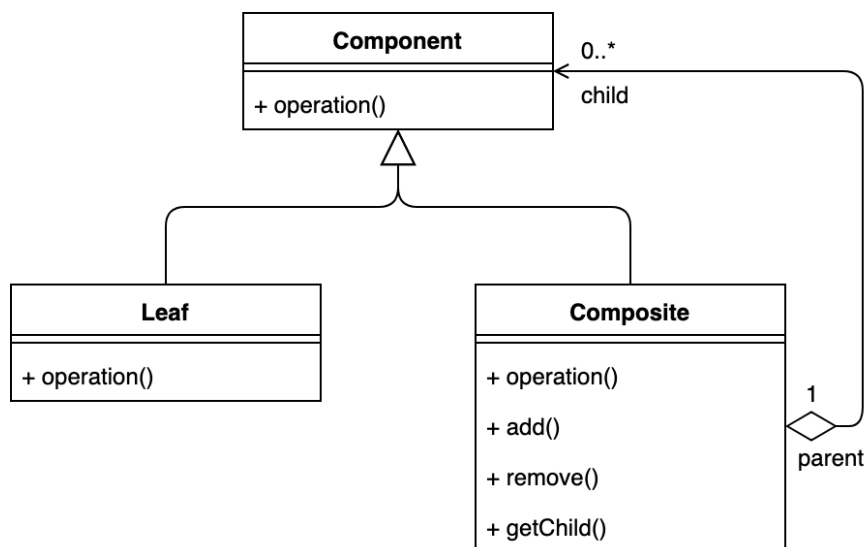
Obrázek 38: Konrekní návrhový vzor observer

### 7.3.2 Composite

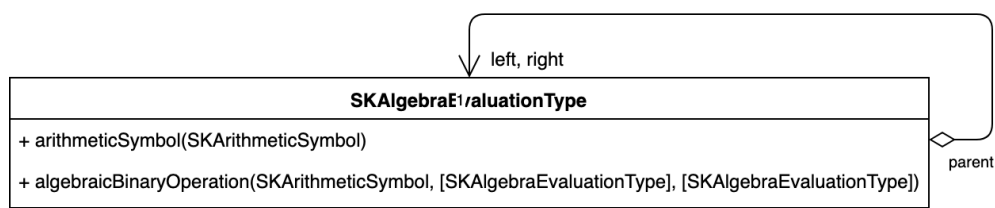
Tento vzor je určený k řešení problémů, kde je potřeba pracovat s hierarchickou strukturou [40]. Obecný princip composit vzoru je takový, že je jeden obecný rodič (component), ze kterého dědí buď už libovolný konečný potomek (leaf) a nebo dědí z něho potomek, který může opět v sobě obsahovat objekty typu component. Tímto způsobem jde se strukturou pracovat jedním způsobem, díky společnému předkovi (Obrázek 39).

Při vývoji tento vzor byl použit na více místech. Jeden z nich je implementace stromové struktury. Další zajímavější příklad z implementace je použití vzoru při převádění aritmetického výrazu, který je možné použít při práci s maticemi, na strukturu, se kterou je možné dále pracovat, respektive strukturu, která se předá syntaktickému analyzátoru. Jelikož výraz může obsahovat v sobě opět další výraz, například v dělení je číselník i jmenovatel opět aritmetický výraz a samotné dělení taky. U obrázku (40), který popisuje konkrétní použití vzoru, je potřeba

dodat, že se jedná o výčtový typ, který je v programovacím jazyce Swift více flexibilní. V tomto případě to znamená, že zde jednotlivé případy výčtového typu mohou obsahovat libovolné data. true



Obrázek 39: Obecný návrhový vzor composite

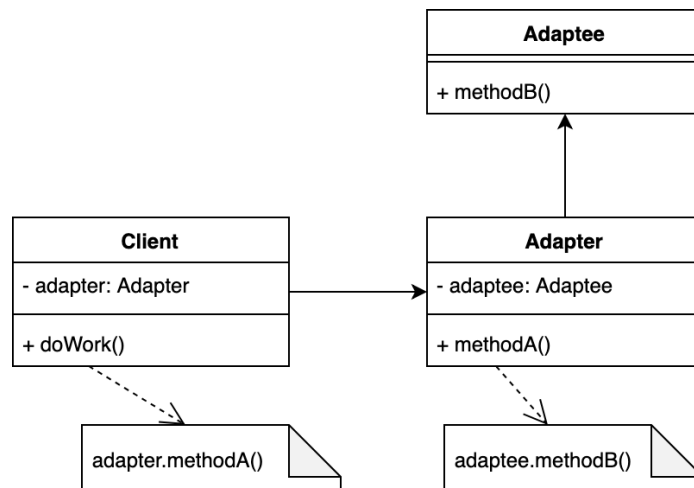


Obrázek 40: Konkrétní návrhový vzor composite

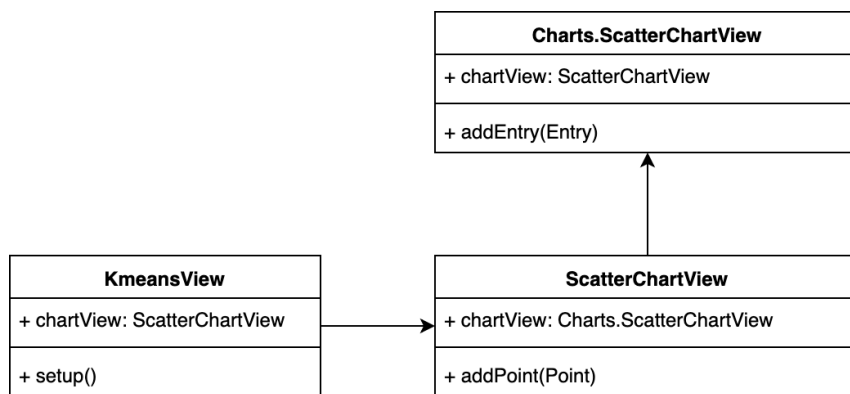
### 7.3.3 Adapter

V mnoha situacích je potřeba zaobalit nějaké rozhraní [39]. Důvodů může být mnoho, například složitá práce s existujícím rozhraní, nebo pouze zaobalit rozhraní z knihovny od třetí strany a tím se vyhnout v budoucnu komplikacím v celém projektu při změně knihovny. Vzor adapter je k tomu určený, jelikož zaobaluje existující rozhraní a vystavuje ho pod nové rozhraní (Obrázek 41).

Tento vzor byl použit při práci s knihovnou od třetí strany, kde došlo k zaobalení jejího rozhraní na snadněji použitelné pro tento projekt (Obrázek 42).



Obrázek 41: Obecný návrhový vzor adapter

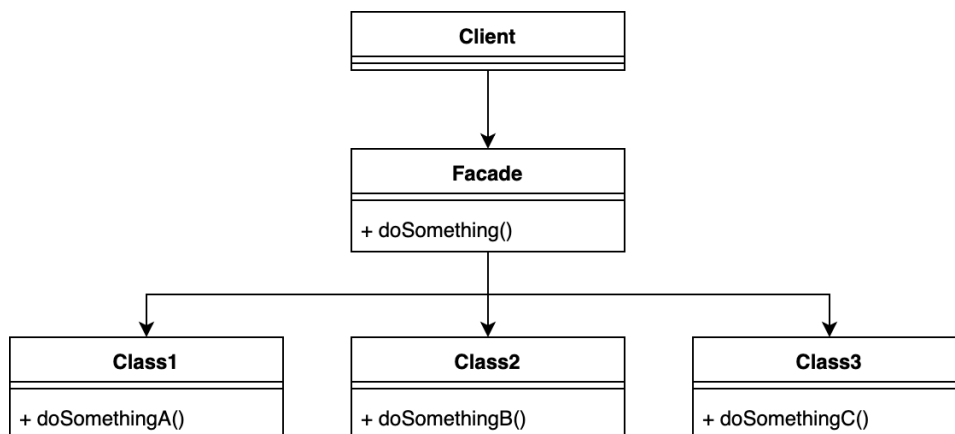


Obrázek 42: Konkretní návrhový vzor adapter

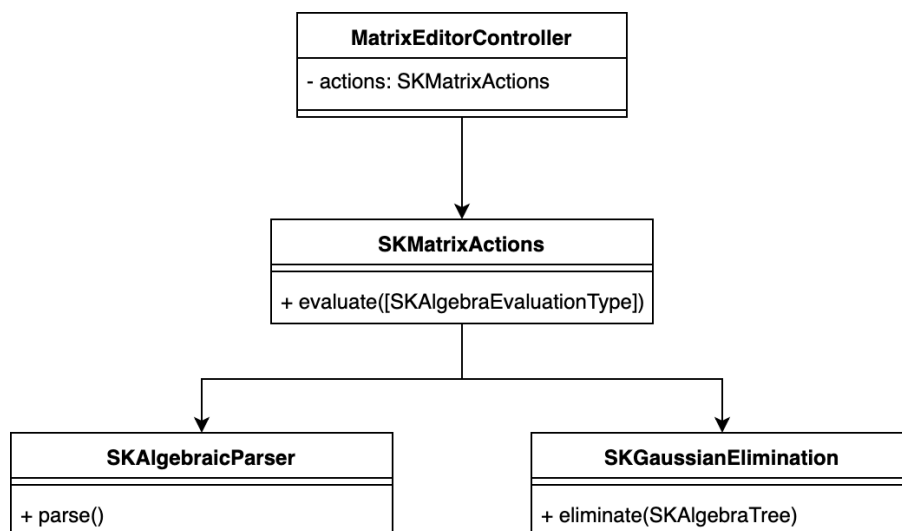
#### 7.3.4 Facade

Vzor facade, je určený k zjednodušení použití subsystému, nebo jiného systému [41]. Princip je takový, že se složité akce zaobalí novým zjednodušeným rozhraním, které provede už jednotlivé akce daného subsystému. Z venku tento systém pak vystupuje pod jednoduchým rozhraním a složitá implementační logika je schována za tímto rozhraním (Obrázek 43).

Tento princip se využívá při komunikaci s výše popsányými dynamickými frameworky. Tyto frameworky obsahují spoustu složité logiky, která je rozdělena na části a vystavena pod jednoduchým rozhraním (Obrázek 44).



Obrázek 43: Obecný návrhový vzor facade

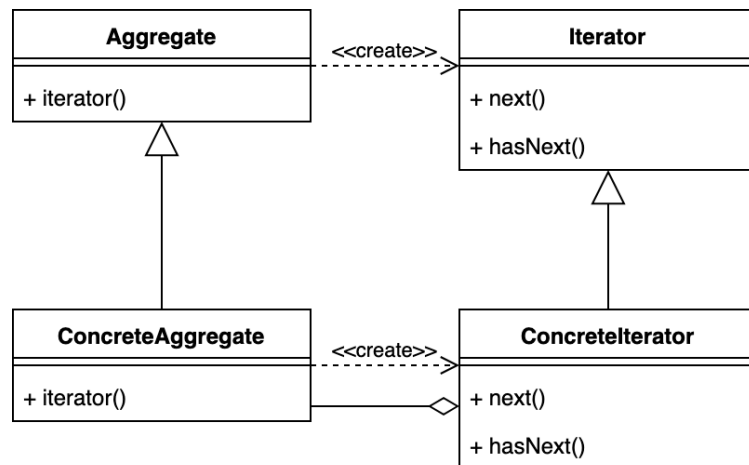


Obrázek 44: Konrekní návrhový vzor facade

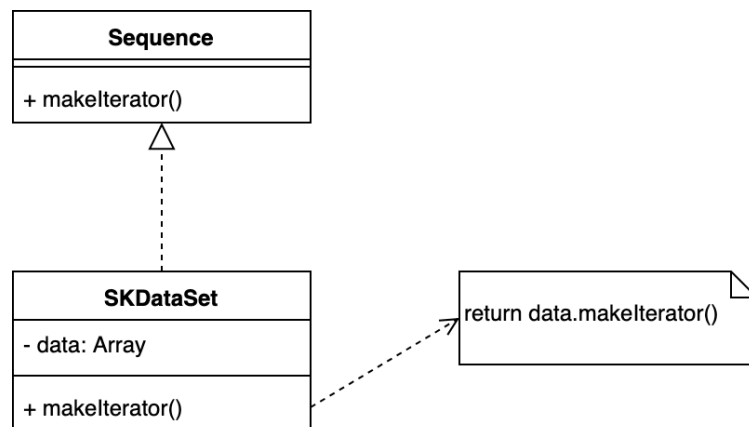
### 7.3.5 Iterator

Iterator je jednoduchý vzor, který slouží k procházení prvků libovolné kolekce nebo objektu, bez znalosti jejich vnitřní implementace [43]. Používá se vždy jednotné jednoduché rozhraní, kde jsou dvě metody. První je `hasNext`, která zjistí jestli je ještě nějaký prvek k dispozici a druhá metoda je `next`, která vrátí prvek (Obrázek 45).

Při implementaci byl použit vzor iterator na vlastní implementaci třídy, která se stará o uchování dat. Konkrétně to bylo použité při K-means algoritmu, který pracuje s datovou sadou, kterou prochází postupně prvek za prvkem. Aby libovolný objekt (třída) mohl vystupovat jako iterator, tak jsou k tomu v programovacím jazyku Swift už předpřipravené rozhraní. Výsledná implementace je potom velmi jednoduchá (Obrázek 46).



Obrázek 45: Obecný návrhový vzor iterator

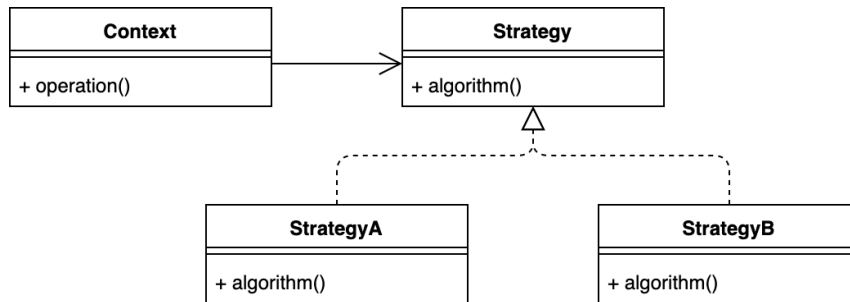


Obrázek 46: Konrekní návrhový vzor iterator

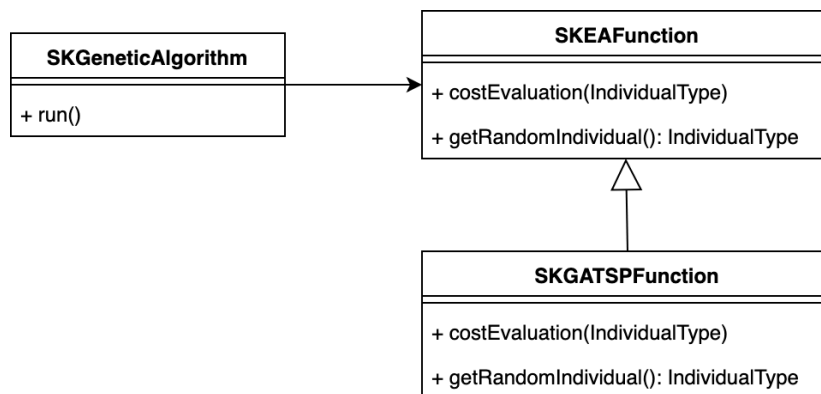
### 7.3.6 Strategy

Jedná se o velmi užitečný návrhový vzor. Strategy slouží k výměně určité části algoritmu při běhu programu [46]. Jinými slovy, když je potřeba konkrétní část algoritmu nahradit za jinou, ale zbytek hlavního algoritmu nechat beze změny. Tím se zle vyhnout zbytečné duplicitě kódu a pouze libovolný úsek algoritmu takto měnit přes strategy vzor (Obrázek 47). Příkladem může být výpočet vzdálenosti mezi body v prostoru, kde tento výsledek pak program použije pro další výpočty. Metrik je mnoho a mnohdy je potřeba použít jiné, než třeba klasickou eukleidovskou vzdálenost.

V implementaci byl tento vzor použit na více místech, například: K-means a genetický algoritmus. U K-means dochází k výměně algoritmu pro výpočet vzdálenosti mezi body, na základě metriky. U genetického algoritmu se jedná o výpočet účelové funkce. Jelikož zbytek algoritmu je stejný pro libovolnou účelovou funkci (Obrázek 48).



Obrázek 47: Obecný návrhový vzor strategy

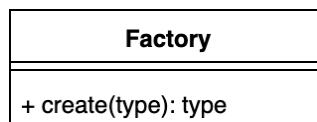


Obrázek 48: Konrekní návrhový vzor strategy

### 7.3.7 Factory Method

Factory Method je velmi jednoduchý vzor [42]. Často je potřeba vytvářet objekty podle nějakého typu nebo podmínky, které mají ale společné rozhraní nebo předka. Metoda vytvoří konkrétní objekt, který je vrácen z metody jako dané rozhraní nebo předek (Obrázek 49).

Tento vzor byl použit při vytváření pracovních ploch, respektive když si uživatel vybírá, která pracovní plocha se mu zobrazí. Tyto plochy mají definované jednotné rozhraní přes které se s pracovní plochou komunikuje, ale už je jedno o jakou se konkrétně jedná (Obrázek 50).



Obrázek 49: Obecný návrhový vzor factory method

<b>PlaygroundsFactory</b>
+ getEditorController(ToolType): PlaygroundEditorController

Obrázek 50: Konrekní návrhový vzor factory method

### 7.3.8 Singleton

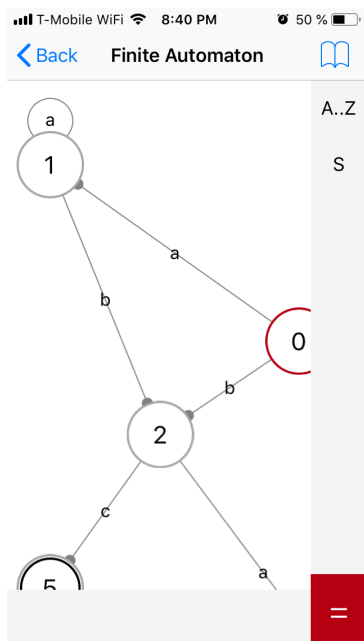
Na tento vzor je mnoho rozdílných názorů. Někteří ho neuznávají a druzí zase ano. Jedná se o vzor, který zajišťuje vytvoření pouze jedné instance při celém běhu programu [45]. A to tak, že dojde k vytvoření sama sebe přes statickou metodu a konstruktor je z venku nepřístupný (Obrázek 51). Vzor tímto částečně porušuje objektově orientované praktiky. Nicméně ať už jsou názory jakékoliv, tak co se týká praxe na iOS, tak Apple v pár částech svého systému tento vzor používá, takže při komunikaci se systémem dochází k jeho využívání.

<b>Singleton</b>
- instance: Singleton
- Singleton() + getInstance()

Obrázek 51: Obecný návrhový vzor singleton

## 8 Ukázka hotového řešení

V této kapitole následuje ukázka finálního systému, který běží na iPhone. Jedná se o ukázky pracovní plochy pro různé nástroje. Ukázky zobrazují konečný automat (Obrázek 52), matice (Obrázek 53), bezkontextové gramatiky (Obrázek 55) a genetický algoritmus (Obrázek 54).

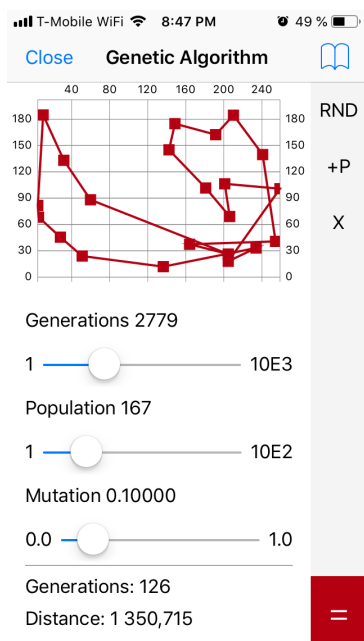


Obrázek 52: Ukázka - konečný automat

$$A = \begin{bmatrix} 1 + \frac{1}{12} & 1 & 1 \\ 1 & 16 & 1 \\ 5 & 14 & 3 \end{bmatrix} + B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Obrázek 53: Ukázka - matice



Obrázek 54: Ukázka - genetický algoritmus

S -> A|a  
A -> Sa|ε  
B -> bb|e  
R -> D|c  
D -> R|a  
S -> Sa|a

Obrázek 55: Ukázka - bezkontextové gramatiky



## 9 Závěr

Cílem práce bylo vytvoření systému pro výuku vybraných partií matematiky a teoretické informatiky. Práce je zaměřena na analýzu dat, kde vstupní data generuje uživatel, který systém používá. Během vývoje systému, byly použity znalosti z matematiky pro zpracování dat, strojového učení, datových struktur a algoritmizace úloh.

V analýze současného stavu řešení byly identifikovány hlavní nedostatky. Z výsledků analýzy a vlastních požadavků na nově vytvářený systém byly sesbírány případy užití a funkční požadavky. Mezi hlavní nedostatky, které byly nalezeny v existujících řešeních, patří složité uživatelské rozhraní. V mnoha případech velmi špatné nebo složité uživatelské rozhraní vedlo k tomu, že takový software nebyl použitelný. Mezi další nedostatky lze zařadit vždy velmi podobné zaměření softwaru na vybrané části algebry. Existující software pro ostatní vědní disciplíny, například teoretická informatika, se zde vůbec nevyskytuje. Tento problém byl jeden z klíčových požadavků na nový systém, kterým se tato práce zabývala.

Nově navrhnuty systém obsahuje možnost pracovat s maticemi, regulárními jazyky, konečnými automaty, bezkontextovou gramatikou a navíc přidává základní algoritmus K-means ze strojového učení a z oblasti biologicky inspirovaných algoritmů se zde nachází genetický algoritmus. Pro každou vyjmenovanou oblast má uživatel možnost zadat vlastní vstup, se kterým lze provádět akce podle daného kontextu. Pro zadávání uživatelských vstupů bylo vytvořeno unikátní rozhraní, které se mění podle vybrané oblasti, se kterou uživatel pracuje. Jinými slovy, jestli chce uživatel pracovat s konečnými automaty, tak není zde důvod mít viditelné vstupy pro práci s maticemi a naopak. Celé rozhraní pro zadávání vstupů má vždy stejnou strukturu. Uživatel tak vždy ví, kde má co hledat, bez ohledu na to, v jakém kontextu zrovna pracuje (matice, konečné automaty, bezkontextové gramatiky, atd.). Díky tomu byla dodržena podmínka na jednoduché a přehledné rozhraní, která byla identifikována jako klíčová v analýze současného stavu řešení.

Celý systém je rozdělen na tři hlavní části. První část je samotná iOS aplikace, se kterou uživatel interaguje. Tato část obsahuje veškeré uživatelské rozhraní. Druhá část je framework (neboli knihovna), která obsahuje veškeré výpočetní algoritmy, podpůrné algoritmy, datové struktury, parsery a syntaktickou analýzu. Jedná se o hlavní část systému. Framework je do iOS aplikace pouze nalinkován a aplikace využívá jeho služby přes dostupné veřejné programové rozhraní (API frameworku). Takto zvolené řešení umožňuje využít tento framework, respektive veškeré algoritmy, i na jiných Apple platformách než pouze iOS systém, například na desktopovém operačním systému macOS. Poslední část je server, neboli backend, který poskytuje a přijímá data skrze REST API. Zde server slouží pouze pro správu uživatelů v systému a jejich metadat.

Před začátkem implementace byly porovnány základní architektury a identifikovány jejich slabiny a přednosti. Podle požadavků na vytvářený systém byla zvolena nejvhodnější architektura,

konkrétně MVVM. V implementační části došlo k dodržení všech definovaných podmínek a funkčních požadavků. Ať už se to týká architektury systému, nebo možnost vytvořit systém tak, aby byl velmi dobře udržovatelný a v budoucnu dále rozšiřitelný o nové oblasti. V implementaci se také nachází syntaktická analýza uživatelských vstupů, která zajišťuje, že vstup odpovídá definované bezkontextové gramatice. Při implementaci systému, ať už z pohledu návrhu architektury, nebo jednotlivých menších podproblémů, byly použité správné praktiky programování, jako například celá řada návrhových vzorů (observer, composite, facade, adapter, ...). Všechny zásadní algoritmy, které provádí matematické výpočty, operace, vykreslování do uživatelského rozhraní a ostatní podpůrné algoritmy, byly implementovány manuálně bez použití externích knihoven. Celý výsledný systém proto obsahuje více než 12 000 řádků kódu.

Výsledný systém splnil požadavky definované na začátku práce. Systém je dobře strukturovaný a připraven na další vývoj nových oblastí jak z matematiky tak i ostatních vědeckých oblastí. Systém bude nasazen do praxe a bude se studovat zpětná vazba od reálných uživatelů. Na základě zpětné vazby bude probíhat následující vývoj tak, aby došlo k uchycení produktu mezi ostatními existujícími produkty podobného zaměření.

## Literatura

- [1] 2019 Bohemian B.V. *Sketch - The digital design toolkit* [online]. c2019, [cit. 2019-02-14]. Dostupné z: <<https://www.mathworks.com/>>
- [2] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Přeložil Bogdan KISZKA. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.
- [3] AlamoFire Software Foundation. Inc. *AlamoFire/AlamoFire: Elegant HTTP Networking in Swift* [online]. c2018, [cit. 2018-12-29]. Dostupné z: <<https://github.com/AlamoFire/AlamoFire>>
- [4] Alfred V. Aho; John E. Hopcroft; Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms* Addison-Wesley, 1974. 157-162 s. ISBN 978-0201000290.
- [5] Antonio Giarrusso. *iMathematics - GCSE Maths Helper and Solver* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<http://www.mobixee.com/>>
- [6] Apple Inc. *Core Data Programming Guide: What Is Core Data?* [online]. c2018, [cit. 2018-12-30]. Dostupné z: <<https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/>>
- [7] Apple Inc. *Metal / Apple Developer Documentation* [online]. c2018, [cit. 2018-12-30]. Dostupné z: <<https://developer.apple.com/documentation/metal>>
- [8] Apple Inc. *Model-View-Controller* [online]. c2012, [cit. 2018-12-19]. Dostupné z: <<https://developer.apple.com/library/archive/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html>>
- [9] Apple Inc. *Overview of Dynamic Libraries* [online]. c2017, [cit. 2018-12-20]. Dostupné z: <<https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/OverviewOfDynamicLibraries.html>>
- [10] Apple Inc. *SpriteKit / Apple Developer Documentation* [online]. c2018, [cit. 2018-12-30]. Dostupné z: <<https://developer.apple.com/documentation/spritekit>>
- [11] Apple Inc. *Xcode - Apple Developer* [online]. c2018, [cit. 2018-12-21]. Dostupné z: <<https://developer.apple.com/xcode/>>
- [12] Apple Inc. *iOS 12 - Apple Developer* [online]. c2018, [cit. 2018-12-19]. Dostupné z: <<https://developer.apple.com/ios/>>
- [13] Bitbucket. *Bitbucket / The Git solution for professional teams* [online]. c2018, [cit. 2018-12-21]. Dostupné z: <<https://bitbucket.org/>>

- [14] Brilliant. *Brilliant / Math and science done right* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<https://brilliant.org/>>
- [15] CHAMBERS, John M. *Graphical Methods for Data Analysis: 0. Chapman and Hall/CRC.*, 2017.
- [16] Carthage. *GitHub - Carthage/Carthage: A simple, decentralized dependency manager for Cocoa* [online]. c2018, [cit. 2018-12-21]. Dostupné z: <<https://github.com/Carthage/Carthage>>
- [17] Chegg Inc. *MATH 42* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<http://math-42.com/>>
- [18] Daniel Cohen Gindi, Philipp Jahoda. *danielgindi/Charts: Beautiful charts for iOS/tvOS/OSX! The Apple side of the crossplatform MPAndroidChart.* [online]. c2016, [cit. 2018-12-30]. Dostupné z: <<https://github.com/danielgindi/Charts>>
- [19] Datta Kanti Bhushan. *Matrix and Linear Algebra: Aided with MATLAB.* PHI Learning, 2008. ISBN 978-8120336186.
- [20] Datta Kanti Bhushan. *Matrix and Linear Algebra: Aided with MATLAB* [online]. c2008, [cit. 2018-10-28]. Dostupné z: <[http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL\\_123456789/4037/5/cs516-2.pdf](http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL_123456789/4037/5/cs516-2.pdf)>
- [21] Django Software Foundation. *The Web framework for perfectionists with deadlines / Django* [online]. c2018, [cit. 2019-01-14]. Dostupné z: <<https://www.djangoproject.com/>>
- [22] E. Kowalski. *Linear Algebra* [online]. c2019, [cit. 2019-01-08]. Dostupné z: <[https://people.math.ethz.ch/~kowalski/script-la.pdf\\_patterns/observer](https://people.math.ethz.ch/~kowalski/script-la.pdf_patterns/observer)>
- [23] Euclides Inc. *FX Solvers helps your math problem solving in Algebra and Calculus* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<http://www.euclides.com/>>
- [24] Fabric *Fabric - App Development Platform for teams* [online]. c2018, [cit. 2018-12-30]. Dostupné z: <<https://get.fabric.io/>>
- [25] Harry H. Porter. *Converting an NFA to a DFA* [online]. c2015, [cit. 2018-09-08]. Dostupné z: <<http://web.cecs.pdx.edu/~harry/compilers/slides/LexicalPart3.pdf>>
- [26] Ivan Zelinka. *Biologicky inspirované výpočty* [online]. c2008, [cit. 2018-10-28]. Dostupné z: <<http://dataanalysis.vsb.cz/data/Vyuka/BIA/BIV-AUI.pdf>>
- [27] J. Richard Büchi. *Finite Automata, Their Algebras and Grammars: Towards a Theory of Formal Expressions* Springer, 2012. ISBN 9781461388555.
- [28] JetBrains s.r.o. *PyCharm* [online]. c2018, [cit. 2018-12-21]. Dostupné z: <<https://www.jetbrains.com/pycharm/>>

- [29] Khan Academy. *Khan Academy / Free Online Courses, Lessons and Practice* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<https://www.khanacademy.org/>>
- [30] Kirill Fakhroutdinov. *Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc.* [online]. c2019, [cit. 2019-02-26]. Dostupné z: <<https://www.uml-diagrams.org/>>
- [31] MELOUN, Milan a Jiří MILITKÝ. *Kompendium statistického zpracování dat*. Vyd. 3. Praha: Karolinum, 2012. ISBN 978-80-246-2196-8.
- [32] Mathway. *Mathway / Algebra Problem Solver* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<https://www.mathway.com/>>
- [33] Miroslav Beneš. *Překladače* [online]. c2018, [cit. 2018-10-18]. Dostupné z: <<http://www.cs.vsb.cz/behalek/vyuka/pjp/skripta/skr-mb.pdf>>
- [34] Nils J. Nilsson. *Introduction to Machine Learning* [online]. c1996, [cit. 2018-10-28]. Dostupné z: <[http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL\\_123456789/4037/5/cs516-2.pdf](http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL_123456789/4037/5/cs516-2.pdf)>
- [35] Objc.io. *Introduction to MVVM · objc.io* [online]. c2014, [cit. 2018-12-19]. Dostupné z: <<https://www.objc.io/issues/13-architecture/mvvm/>>
- [36] Photomath Inc. *Photomath - Scan. Solve. Learn.* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <<https://photomath.net/>>
- [37] ReactiveX. *ReactiveX/RxSwift: Reactive Programming in Swift* [online]. c2018, [cit. 2018-12-29]. Dostupné z: <<https://github.com/ReactiveX/RxSwift>>
- [38] Richard J. Trudeau. *Introduction to Graph Theory (Dover Books on Mathematics)* Dover Publications, 1994. ISBN 9780486678702.
- [39] SourceMaking.com. *Adapter Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: <[https://sourcemaking.com/design\\_patterns/adapter](https://sourcemaking.com/design_patterns/adapter)>
- [40] SourceMaking.com. *Composite Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: <[https://sourcemaking.com/design\\_patterns/composite](https://sourcemaking.com/design_patterns/composite)>
- [41] SourceMaking.com. *Facade Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: <[https://sourcemaking.com/design\\_patterns/facade](https://sourcemaking.com/design_patterns/facade)>
- [42] SourceMaking.com. *Factory Method Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: <[https://sourcemaking.com/design\\_patterns/factory\\_method](https://sourcemaking.com/design_patterns/factory_method)>
- [43] SourceMaking.com. *Iterator Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: <[https://sourcemaking.com/design\\_patterns/iterator](https://sourcemaking.com/design_patterns/iterator)>

- [44] SourceMaking.com. *Observer Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: [https://sourcemaking.com/design\\_patterns/observer](https://sourcemaking.com/design_patterns/observer)
- [45] SourceMaking.com. *Singleton Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: [https://sourcemaking.com/design\\_patterns/singleton](https://sourcemaking.com/design_patterns/singleton)
- [46] SourceMaking.com. *Strategy Design Pattern* [online]. c2019, [cit. 2019-01-21]. Dostupné z: [https://sourcemaking.com/design\\_patterns/strategy](https://sourcemaking.com/design_patterns/strategy)
- [47] Stephen G. Kobourov. *Force-Directed Algorithms* [online]. c2013, [cit. 2018-10-05]. Dostupné z: [https://www.researchgate.net/publication/304840569\\_Force-Directed\\_Algorithms](https://www.researchgate.net/publication/304840569_Force-Directed_Algorithms)
- [48] Swiftgen. *SwiftGen/SwiftGen: The Swift code generator for your assets, storyboards, Localizable.strings, ... — Get rid of all String-based APIs!* [online]. c2018, [cit. 2018-12-21]. Dostupné z: <https://github.com/SwiftGen/SwiftGen>
- [49] Symbolab. *Symbolab Math Solver - Step by Step calculator* [online]. c2019, [cit. 2019-02-02]. Dostupné z: <https://www.symbolab.com/>
- [50] The MathWorks Inc. *MathWorks - Makers of MATLAB and Simulink - MATLAB and Simulink* [online]. c2019, [cit. 2019-02-03]. Dostupné z: <https://www.mathworks.com/>
- [51] VÁVRŮ, Jiří. *iPhone: vývoj aplikací*. Praha: Grada Publishing, 2012. ISBN 978-80-247-4457-5.
- [52] Wikimedia Foundation, Inc. *Gaussian elimination - Wikipedia* [online]. c2018, [cit. 2018-09-08]. Dostupné z: [https://en.wikipedia.org/wiki/Gaussian\\_elimination/](https://en.wikipedia.org/wiki/Gaussian_elimination/)
- [53] Wolfram Research, Inc. *Cellular Automaton – from Wolfram MathWorld* [online]. c2019, [cit. 2019-01-18]. Dostupné z: <http://mathworld.wolfram.com/CellularAutomaton.html>
- [54] Wolfram. *Wolfram: Computation Meets Knowledge* [online]. c2019, [cit. 2019-02-03]. Dostupné z: <https://www.wolfram.com/>
- [55] Wolfram. *Wolfram/Alpha: Computational Intelligence* [online]. c2019, [cit. 2019-02-03]. Dostupné z: <https://www.wolframalpha.com/>
- [56] ZAKI, Mohammed J.; MEIRA JR, Wagner; MEIRA, Wagner. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

## Seznam příloh

- Příloha A - Příloha v IS EDISON.

## 10 Přílohy

### 10.1 Příloha A

#### 10.1.1 Příloha v IS EDISON

```
/
├── source.....adresář se zdrojovými soubory
│   ├── code.....adresář obsahující implementaci
│   └── thesislutex.tex.....zdrojový soubor práce v TEX
└── texts.....texty práce
    └── thesis.pdf ..... diplomová práce ve formátu PDF
```